Requirements of the SALTY project

Philippe Collet, M.-A. Abchir, Thierry Bathias, Mireille Blay-Fornarino, Filip Krikava, Julien Lenoir, Julien Lesbegueries, Sébastien Madelénat, Jacques Malenfant, David Manset, et al.

https://salty.unice.fr/

ANR SALTY

Self-Adaptive very Large disTributed sYstems

Work Package:	WP1 - Requirements and Architecture
Coordinator:	UNS
Deliverable:	D-1.1
Title:	**Requirements of the SALTY project**
Submission date:	2^{nd} August 2010
Project start date:	1^{st} November 2009, duration: 36 months
Revision:	201
Last change:	30.07.2010

Authors

(including authors of appendices, see appendices for detailed authorship)

Author	**Affiliation**	**Role**
P. Collet	UNS	Lead
M.-A. Abchir	Deveryware & Univ. Paris 8	Writer
T. Bathias	Deveryware	Writer
M. Blay-Fornarino	UNS	Writer
F. Křikava	UNS	Writer
J. Lenoir	Thales	Writer
J. Lesbegueries	EBM Petals Link	Writer
S. Madelénat	Thales	Writer
J. Malenfant	UPMC	Writer
D. Manset	MAAT-G	Writer
O. Melekhova	UPMC	Writer
R. Mollon	MAAT-G	Writer
J. Montagnat	UNS	Writer
R. Nzekwa	INRIA Lille	Writer
A. Pappa	Univ. Paris 8	Writer
J. Revillard	MAAT-G	Writer
R. Rouvoy	INRIA Lille	Writer
L. Seinturier	INRIA Lille	Writer
I. Truck	Univ. Paris 8	Writer

Contents

CHAPTER 1

Introduction

This document is the first external deliverable of the SALTY project (Self-Adaptive very Large disTributed sYstems), funded by the ANR under contract ANR-09-SEGI-012. It is the result of task 1.1 of the Work Package (WP) 1 : *Requirements and Architecture*.

Its objective is to identify and collect requirements from use cases that are going to be developed in WP 4 (*Use cases and Validation*). Based on the study and classification of the use cases, requirements against the envisaged framework are then determined and organized in *features*. These features will aim at guide and control the advances in all work packages of the project. As a start, features are classified, briefly described and related scenarios in the defined use cases are pinpointed. In the following tasks and deliverables, these features will facilitate design by assigning priorities to them and defining success criteria at a finer grain as the project progresses.

This report, as the first external document, has no dependency to any other external documents and serves as a reference to future external documents. As it has been built from the use cases studies that have been synthesized in two internal documents of the project, extracts from the two documents are made available as appendices (cf. appendices B and C).

1.1 Overall Context and Motivations

The growing complexity of software led to huge costs in distributing it to end-users and maintaining it. Service Oriented Architectures (SOA) intend to cut down the complexities and costs of software. As most companies forecast to make larger use of these technologies at all levels of the software ecosystems, mastering the resulting distributed infrastructure at a very large scale is a crucial need. The SALTY project aims at proposing a new step ahead regarding run-time self-adaptation of very large scale distributed systems. These adaptations are typically triggered in response to variations of performance required by the applications or to hardly predictable events (software faults, hardware failures, mobility, etc.). All of these have an impact onto the systems resources availability (memory, processor speed, network bandwidth, etc.).

A lot of work towards software and/or hardware based self-adaptation has been carried out and deployed into the field, with resource reservations in telecoms, task scheduling in operating systems, redundant infrastructures in safety critical applications (*e.g.*, transportation, nuclear power plants) and even the Internet that provides reasonable world-wide network availability. But, none of these solutions is designed to address successfully run-time self-adaptation of large scale distributed systems, which are consistent systems based on distributed hardware platforms connected through heterogeneous networks, operating distributed middleware, and supporting collaborating applications. This challenge is now usually put under the umbrella of *Autonomic Computing*, which aims at building software systems in such a way that they are self-managing. Autonomic computing is a broad research domain, covering a large spectrum of areas (software modeling, reflective models, decision-making, large-scale coordination, etc.). It is very clear that there is a huge gap to be filled in order to get very large scale distributed systems

able to autonomously decide for adaptations at local and/or global scale, taking into account trade-offs between cost, performance (sometimes with real-time constraints) and availability.

1.2 Definitions

This section gathers several definitions that are relevant to the SALTY project, ranging from terms related to autonomic computing to followed approaches and studied use cases. It must be noted that the definitions related to the project is maintained on the internal collaborative web site of the project.

Adaptation: process by which a software system of application is modified during its execution to match the changes in the current requirements of its users and the current state of its environment.

Automatic control: the mathematical and engineering theories, models and methods used to design and implement mechanized control systems that regulate themselves by acting upon actuators in order to achieve an objective for the controlled system.

Autonomic computing: self-managing computing model named after, and patterned on, the human body's autonomic nervous system. An autonomic computing system would control the functioning of computer applications and *systems* without input from the user, in the same way that the autonomic nervous system regulates body systems without conscious input from the individual. The goal of autonomic computing is to create *systems* that run themselves, capable of high-level functioning while keeping the system's complexity invisible to the user. Details on an implementation of Autonomic Computing following the MAPE-K principles are given in section 1.3.

Cell-Id: a geo-positioning technique that consists in identifying the cell in which a device connected to a GSM network is, and to approximate its position given the geographical locus of this cell. This positioning technique does not involve specific intervention from the cell-phone or device itself; it is rather a paid-for service provided by the GSM operator using its infrastructure that continuously tracks the current cell for all cell phones in order to pass on the calls.

Complex Event Processing: Complex Event Processing (CEP) is the use of technology to predict high-level events likely to result from specific sets of low-level factors. CEP identifies and analyzes cause-and-effect relationships among events in real-time, allowing personnel to pro-actively take effective actions in response to specific scenarios. CEP is an evolving paradigm originally conceived in the 1990s by Dr. David Luckham at Stanford University [10].

Component-Based System (CBS): A component-based system relies on two important topics : **Component** and *System*. According to the system, the component concept corresponds to software parts [13] or physical parts of a system [3]. In any case, components enable practical reuse of system parts. In a more specific way, "a software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition." [13]. A CBS corresponds

consequently to an assembly of components. The component-based systems approach potentially overcomes difficulties associated with developing and maintaining monolithic software applications and increases capability to accommodate change.

Deployment: General deployment process consists of several interrelated activities (release, install and activate, deactivate, adapt, update, built-in, version tracking, uninstall, retire) with possible transitions between them. These activities can occur at the producer site or at the consumer site or both. Because every software system is unique, the precise processes or procedures within each activity can hardly be defined. Therefore, "deployment" should be interpreted as a general process that has to be customized according to specific requirements or characteristics.

Feedback control loop: kind of *automatic control* system where the controller makes repetitive actions upon actuators using information obtained from the controlled system to tailor its actions to the current state of the latter. Details on the MAPE-K organization of the feedback control loop are given in section 1.3.

Framework: "A framework is the skeleton of an application that can be customized by an application developer" [7].

Framework SALTY: The *SALTY framework* aims at being a customizable and reusable skeleton of "*autonomic*" *feedback control loops*, targeting *large-scale* distributed *systems*. The *framework* rests on SOA and SCA infrastructures and is intended to be built using *Model-Driven Engineering* techniques.

Fuzzy logic: is an extension of classical logic meant to reason about imprecise data. In fuzzy logic, the binary truth values are replaced by a continuous domain $[0, 1]$ of degree of truth. Based on degrees of membership expressed as membership functions defining fuzzy subsets, fuzzy logic use several variants of generalized modus ponens as its primary inference rule. Fuzzy logic emerged as a consequence of the 1965 proposal of fuzzy set theory by Lotfi Zadeh [14].

Geotracking: the process by which positioning techniques based on different types of devices are use to follow mobile entities over a geographical area in order to fulfill some objective. Geotracking therefore involves two convergent actions repeated continuously: getting the position of the mobile and relating this position to the geographical locus to which it belongs.

Global Positioning System: the global positioning system describes in the general sense a positioning system involving a constellation of satellites orbiting the earth emitting continuously signals from which dedicated devices (namely GPS) can compute their current position by triangulation. Devices need to receive signals from at least three different satellites in order to be able to locate itself. More satellites add precision to the computation, as it is also sensible to the angle between the different emitters.

Grid computing: Grid computing is a term referring to the combination of computer resources from multiple administrative domains to reach common goal. What distinguishes grid computing from conventional high performance computing systems such as cluster computing is that grids tend to be more loosely coupled, heterogeneous, and geographically dispersed. It is also true that while a grid may be

dedicated to a specialized application, a single grid may be used for many different purposes.

Image Processing: Image processing is any form of signal processing for which the input is an image, such as a photograph or video frame; the output of image processing may be either an image or, a set of characteristics or parameters related to the image.

Large-Scale System: Within the SALTY framework, we consider that Large-scale systems are *systems* of unprecedented scale in some of these dimensions (extracted from [12], which defines them as "Ultra Large-Scale Systems"):

- amount of data stored, accessed, manipulated, and refined
- number of connections and interdependencies
- number of computational elements
- number of system purposes and user perception of these purposes
- number of (overlapping) policy domains and enforceable mechanisms

"These systems are necessarily decentralized in a variety of ways, developed and used by a wide variety of stakeholders with conflicting needs, evolving continuously, and constructed from heterogeneous parts." [12]) .

Markovian Decision Processes (MDP): a Markovian Decision Process (MDP) is a mathematical framework for a kind of *sequential decision making problem* where the outcomes and transitions among states given a decision are partly random and partly under the control of the decision maker through his sequence of decisions. MDPs are used to model a wide-spectrum of optimization problems solved either via dynamic programming or reinforcement learning.

Model-Driven Engineering (MDE): According to OMG, "model-driven" stands for "a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification" [11].

Monitoring: Monitoring is the regular observation and recording of activities taking place in a system or an application. It is a process of routinely gathering information on all aspects of the system. To monitor is to check on how system activities are progressing. Monitoring also involves giving feedback about the progress of the system to the users, administrators and owners of the system. Reporting enables the gathered information to be used in making decisions for repairing or adapting the system, as well as for improving its performance.

Multi-criteria Decision Making: kind of decision making where several possibly contradictory criteria are used to assess the decisions made and the system on which these decisions are implemented. In the context of SALTY, quality of service is a typical multi-criteria context, as it usually encompasses several different dimensions (latency, availability, performance, precision, etc.). Multi-criteria decision making leverages utility theory and other models of user *preferences* in order to build a non-ambiguous comparison scale among the different criteria.

Preference modeling: elicitation process by which users deal with several possibly contradictory criteria by expressing their preference among them. Arising in the context of *multi-criteria decision-making*, preference modeling aims at putting tuples of values from the different criteria under a unique comparison scale, in order to get

a complete order among them. Several models for preference have been proposed in the past, some of them leveraging properties of preferences, like the general additive independence, to express graphically local dependencies among otherwise independent criteria. GAI-networks and the family of CP-nets formalisms fall into this category. Other approaches construct utility functions directly, mapping tuples to utility values in $\mathbb{R}$, from the tuples themselves and inputs from domain experts (users) who determine their preferences among them, often two by two. These approaches therefore strive to infer the overall complete order from the set of two-by-two orders between tuples given by experts.

Quality of Service (QoS): The term of Quality of Service (QoS) originated from the computer networking domain in which it characterized properties of the networks to deliver service as data flow in a predictable way. QoS attributes often include availability (uptime), throughput (bandwidth), latency (delay), and error rate. In a *SOA* context, QoS usually refers to non-functional properties of services (*e.g.*, Web Services), such as availability (possible immediate usage), accessibility (capacity of serving a request), integrity (capacity to maintain the correction of several interactions), performance (throughput and latency), reliability (maintaining service quality), regulatory (conformance to rules and standards), security (confidentiality and non-repudiation). QoS dimensions are usually used inside *SLA*.

Self-adaptation: process by which a software system or application act upon itself to perform its own *adaptation*. Self-adaptation requires that the software has a reflective architecture in order to be able to perform introspective (examining itself) and intercession (modifying itself). A classification concerning self-adaptive systems is given in section 1.3.

Sequential Decision Making Problem: a sequential decision making problem occurs in decision-making situations where decisions are made over time and trigger changes in the state on the system upon which these decisions are implemented, so that the nature and the outcome of further decisions depend upon prior ones. Hence, sequential decision making usually incurs a trade-off between the short-term outcomes of the next decision and the long-term overall outcomes of the whole sequence of future decisions.

Service Component Architecture (SCA): Relatively new initiative advocated by major software vendors. Its proponents claim it is more natively suited for the delivery of applications that conform with the *SOA* principles. As such, SCA components are supposedly more technologically agnostic. The value proposition of SCA, therefore, is to offer the flexibility for true composite applications, flexibly incorporating reusable components in an *SOA* programming style. The overhead of business logic programmer concerns regarding platforms, infrastructure, plumbing, policies and protocols are removed, enabling a high degree of programmer productivity.

Service-Level Agreement (SLA): A Service-Level Agreement (SLA) is a representation of all features a user (human or machine) should expect to receive by a service. These features encompass both functionality delivered but also the quality experienced by the user, referred as *Quality of Service*. The SLA is usually a negotiated agreement between the consumer and the provider of the service. The agreement usually contains the specified level of service and *QoS*, some means to measure or monitor it, as well as penalty provisions and remedial actions in case of failure. It

is sometimes referred as **service level contract**. In the *SOA* context, specification standards have moved from WSLA to Ws-Agreement (Web Services Agreement Specification).

Service-Oriented Architecture (SOA): This architecture gathers design principles for systems dealing with services integration. A deployed SOA is composed of services that are organized in a loosely coupled way. Services are associated each other thanks to orchestration. This concept consists in a tool able to compose services in a structured way and build executable processes. Main implementation standards associated with SOA are the "Web Services Description Language" (WSDL) for describing services and the "Business Process Executable Language" (BPEL) for orchestrating them.

Stability: the property of a feedback control loop to maintain the system at its nominal state with minimal deviations. Sources of instabilities are: (1) latency in the evolution of the system, when decisions are made before the system reaches its nominal state after a decision, such that a new decision may be mislead by an intermediate state triggering a unnecessary strong decision, (2) power of the control, when the power of available decisions to react to some state forces the system away from the nominal state, thus immediately requiring a corrective action that may again be too strong and so on, and (3) the variations in the input to the system, when they are too large for the power of the control, actions cannot compensate for such large variations that can therefore force the system out of its nominal state at the same pace.

System: We present the SALTY concepts in terms of some existing or planned systems. "That system may include anything: a program, a single computer system, some combination of parts of different systems, a federation of systems, each under separate control, people, an enterprise, a federation of enterprises? A **model** of a system is a description or specification of that system and its environment for some certain purpose." [11]

1.3 Self-Adaptive Systems - Classification and Engineering

This section presents the classification that is used to characterize representative scenarii in SALTY's use cases, and the main principles of the MAPE-K feedback control loop that serves as a general architectural guide for the envisioned framework.

In order to classify and to give elements of comparison between the considered scenarios of all use cases, we choose to rely on a recent classification of self-adaptive systems [5]. This classification defines four modeling dimensions each describing a particular aspect of the system that is relevant to the self-adaptation:

Goals. Goals are the objectives the system under consideration should achieve. They are classified based on their *evolution*, whether they can change within the lifetime of the system; *flexibility* to express the level of uncertainty associated with the goal specification; *duration* to concern the validity of a goal throughout the system's lifetime; *multiplicity* expressing the number of goals associated with the self-adaptability of the system and *dependency* in case the system has multiple goals to capture how they are related to each other.

Change. Changes are the causes of self-adaptation - whenever the system's context changes the system should decide whether it needs to adapt. The context dependable changes of the self-adaptive systems are classified in terms of the *source* (place) where the change has occurred, the *type* and *frequency*, and whether it can be *anticipated*. All these dimensions are important for identifying how the system should act upon a change during run-time.

Mechanisms. The dimension associated with the mechanisms to achieve self-adaptability. This set captures the system reaction towards change, they are related to the adaptation process itself. The dimensions refer to the *type* of self-adaptation that is expected; the level of *autonomy*; *organization* of self-adaptation - centralized, or distributed amongst several components, the impact of self-adaptation in terms of *scope* - local or global; and *duration* - time in which the system is self-adapting (how long does the adaptation last; *timeliness* capturing whether the time period for performing adaptation can be guaranteed; *triggering* identifying whether the change initiating the adaption is triggered by time or event.

Effects. Effects indicates what is the impact of adaptation upon the system. It refers to *criticality* of the adaptation; *predictability* of the consequences of self-adaptation (can be both in value and time), what are the *overhead* associated with it, and whether the system is *resilient*[1] in the face of change.

The full overview about the classification is provided in [5] and a summary in A.

Engineering of these systems is a major challenge. What they have in common is that the design decisions are being moved towards runtime to control dynamic behavior and that an individual system reasons about its state and environment. In both [5, 4] it is strongly argued that self-adaptive systems must be based on the feedback principle taken from control engineering and they advocate that the feedback loops provide the generic mechanism for self-adaptation. In designing self-adaptive systems, the feedback loops that control self-adaptation must become first-class entities.

For a system component to be self-managing, it must have an automated method to collect the details it needs from the system; to analyze those details to determine if something needs to change; to create a plan, or sequence of actions, that specifies the necessary changes and to perform those actions [1]. In the same white paper IBM presents an architectural blueprint for autonomic computing. They introduce an autonomic manager, a component that implements an intelligent control loop - MAPE-K control loop (cf. Fig. 1.1). The name is an abbreviation for *Monitor*, *Analyze*, *Plan*, *Execute* and *Knowledge*. The loop is divided into four parts that share knowledge:

- The *monitor* function provides the mechanisms that collect, aggregate, filter and report details (such as metrics and topologies) collected from a managed resource.
- The *analyze* function provides the mechanisms that correlate and model complex situations (for example, time-series forecasting and queuing models). These mechanisms allow the autonomic manager to learn about the IT environment and help predict future situations.
- The *plan* function provides the mechanisms that construct the actions needed to achieve goals and objectives. The planning mechanism uses policy information to guide its work.

[1] it is related to the persistence of service delivery that can be justifiably trusted, when facing changes.

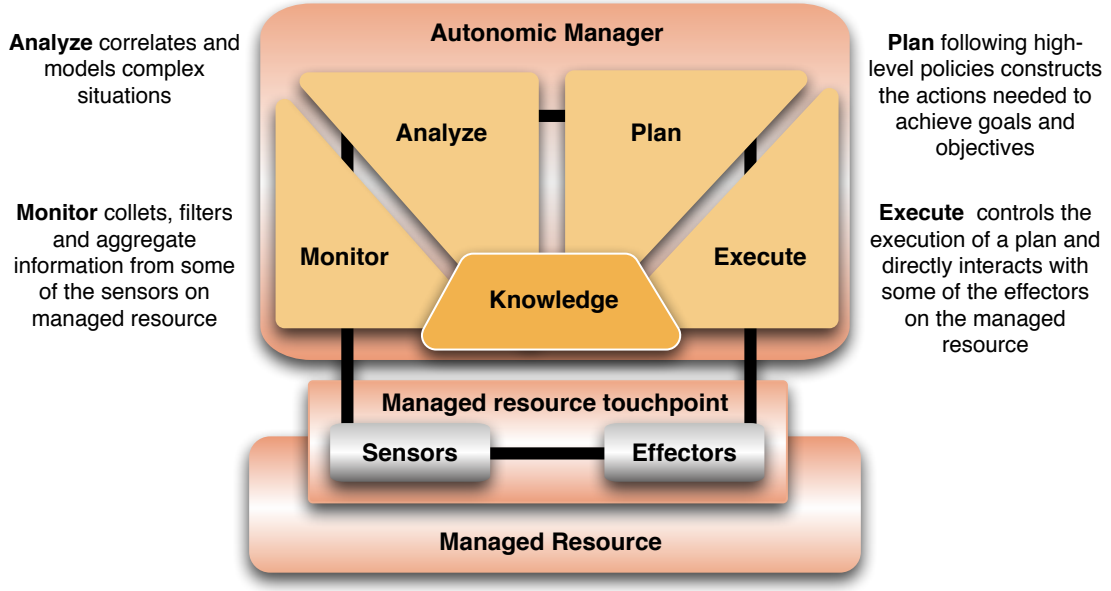


Figure 1.1: MAPE-K control loop

- The *execute* function provides the mechanisms that control the execution of a plan with considerations for dynamic updates.

The autonomic manager provides sensor and effector interfaces for other autonomic managers and components in the distributed infrastructure to use. Using sensors and effectors interfaces for the distributed infrastructure components enables these components to be composed together in a manner that is transparent to the managed resources [1].

1.4 Organization

The remainder of this document is organized as follows. Chapter 2 is briefly describing the two use cases that serve both as requirement sources and validation elements. All representative scenarios are also synthesized and references to complete descriptions in appendices B and C are given. In chapter 3, envisaged features of the SALTY project are categorized and synthetically described. Relations to the previously listed scenarios are also determined. Perspectives and next steps regarding the project are described in chapter 4. They cover analysis and design of first models and running systems making up the SALTY framework, as well as advances in the use cases. Appendix A details the classification that has been previously introduced. In appendix B the scenarii related to the ”Middleware” use cases are detailed. They are decomposed in scenarii regarding the gLite Grid middleware, the desktop fusion middleware (these two being referred as ”Grid”) and the Petals ESB middleware (this one being referred as ”ESB”). Finally, appendix C describes the scenarii that concern the truck tracking application using multi-means geo-positioning.

CHAPTER 2

Use cases

Use cases of the SALTY project strive to evaluate to capability of the SALTY architecture to capture different types of applications and systems, as well as to validate its capability to manage self-adaptations at different level of granularity. Features sought by the SALTY architecture will be presented in more details in the next chapter, but the use cases have been chosen to cover most of them.

The two major use cases of the project concerns on the one hand different middlewares and applications dedicated to large distributed systems, and on the other hand, an application using geotracking. The middleware use cases are presented in the following two sections. Section 2.1 describes the scenarios dedicated to grid management for computation-intensive medical image processing and rendering. In section 2.2, some autonomic scenarios related to a large-scale service bus in the context of SOA are presented. The geotracking scenarios are more precisely related to a logistic applications making use of geotracking. They are described in section 2.3. The two main use cases are first summarized below, and then presented in details, with all scenarios in appendices B and C.

2.1 "Grid" Middleware Use Case

2.1.1 Context

Grid infrastructures have become a critical substrate for supporting scientific computations in many different application areas. Over the last decade, world-wide scale Grids leveraging the Internet capabilities have been progressively deployed and exploited in production by large international consortia. They are grounded on new middleware federating the grid resources and administration frameworks and enabling the proper operation of the global system 24/7. Despite all efforts invested both in software development to achieve reliable middleware and in system operations to deliver high quality of service, grids encounter difficulties to implement the promise of ubiquitous, seamless and transparent computing.

The causes are diverse and rather well identified (complexity of middleware stacks, dependence to many distributed and heterogeneous resources, uncontrolled reliability of the application codes enacted, incompatibilities between software components, difficulty to identify sources of errors, challenging scale of the computing problems tackled, etc.). The practice demonstrates that the human administration cost for grids is high, and end-users are not completely shielded from the system heterogeneity and faults. Heavy-weight operation procedures are implemented by the grid administrators and users have to explicitly deal with unreliability issues [9].

neuGRID

In the SALTY project, the grid use case is related to the neuGRID European infrastructure (`http://www.neugrid.eu`), which aims at supporting the neuroscience community in

carrying out research on neurodegenerative diseases. In neuGRID, the collection of large amounts of imaging data is paired with grid-based computationally intensive data analyses. The infrastructure is developed to run neuro-imaging and data-mining pipelines of algorithms, in particular specializing on Alzheimer's disease. The neuGRID project is the first project within the neuroscientific community to use the Grid technology. Pipelines manipulated in neuGRID are computationally intensive as they enact a mixture of both short and long running I/O demanding algorithms that are applied over large data sets containing tens of thousands of images. It thus brings underlying Grid resources to their limits and highlights technological bottlenecks. neuGRID is utilizing a Grid infrastructure based on the gLite middleware [8].

gLite Middleware

The gLite middleware has been developed as a part of the European project EGEE which delivers a reliable and dependable European Grid infrastructure for e-Science. gLite is architected as a two-levels batch system that federates resources delivered by multiple computing *sites*. Each site is exposing its *Worker Nodes* computing units (WN) through a *Computing Element* (CE) gateway. A high-level meta-scheduler called the *Workload Management System* (WMS) is used as a front end to multiple CEs. Grid applications are sliced in smaller computing *jobs*. Each job is described through a *Job Description Language* (JDL) document. When submitted, a job enters the WMS through a simple web service (WMProxy) and it is passed to the *Workload Manager* (WM) to be queued into a file system-based *Task Queue* (TQ).

neuGRID data challenges

A part of the neuGRID project is a set of validation tests that are run within the infrastructure in order to verify its good performance while meeting user requirements specification. These performance tests are executed in form of *data challenges* in which a very large data set of medical images is analyzed hence putting a lot of stress onto the underlying infrastructure. The second data challenge was running for less than two weeks, and just for comparison, it would have taken couple of years to accomplish the same using a single workstation. During the execution, several failures occur and the most representative has been used to serve as a starting point for identifying scenarios. A power-failure resulting in a whole site (CE) going down, had to be recovered. Submitted jobs were pushed to the other CE where they caused a failure due to an overload. Most of middleware services suffered overload, e.g. the WMS and CE were not able to handle all submitted jobs, a memory leak occurred in one of the WMS subcomponent, etc. Some library incompatibilities occur on job execution despite the dependencies management system of the middleware. Some scenarios presented below rely on this real world experiments.

Desktop Fusion

In addition to the execution middleware, the neuGrid infrastructure provides means for researches to run specialized viewers and to interact with the Grid directly from their desktop. *Desktop Fusion* is an integrated new technology which allows for remote execution of applications. The technology chosen to achieve this is the Open Source version of NX so call FreeNX. The latter provides encrypted and optimized access to remote applications. As desktop Fusion provides users with remote access to applications which may be more of less compute-intensive, several users using *Desktop Fusion* at the same time

could rapidly result in a poor Quality of Service. This problem is typical of data centers or cloud infrastructure, and will be used as a basis for one of the scenarios presented in the next section.

2.1.2 Summary of the scenarii

There are four scenarii related to the gLite middleware and one related to the desktop Fusion infrastructure.

Scenarii on the gLite middleware are the following:

Scenario 1: WMS overload

As the WMS component is the gateway to the gLite job management system, it might get easily overloaded, usually by receiving more requests that it can handle or due to a software problem in the component itself. To deal with this kind of failure, an additional self-healing control loop should be deployed into the infrastructure. This loop interacts with the WMS host's low level operating system probes and periodically monitors resource usage done by WMS process. We define two threshold values for the system. When the first one is reached then WMS is being blocked from all new incoming jobs to be submitted until its resource usage either goes below this threshold or till it reaches the second threshold. If that happens the adaptation mechanism will proceed and restart the WMS. From this basic version, several extensions are expected to be incrementally developed and evaluated: to introduce historical values on resource usage, to make the monitoring of resource usage self-adaptive itself, to consider tolerance zone over the threshold, to make the threshold setting automatic, to manage coordination with other loops developed in the next scenarios, to monitor precisely the effect of the feedback loops, to apply finer grained loops on the WMS subcomponents.

Scenario 2: CE Starvation

During the data challenge run, when the CE had disappeared because of the power failure, the WMS correctly detected the situation and rescheduled all jobs to the other site that remained available. However, the sudden schedule of many jobs resulted in an complete overload on the other site that had to be restarted in the end. This could have been fixed by setting a smaller queue size. Nevertheless, this introduces a different but more severe issue. If the site receiving all rescheduled jobs was not overloaded and continued to work and the other site appeared again, it would have no jobs to execute. This would result into the situation when one site is very busy and the other completely idle, being able to only work on newly arrived jobs. Therefore, in this scenario, the objective is to keep all computing elements optimally utilized and prevent them from both extremes: an overload, due to large number of jobs getting scheduled, on one hand and a starvation, with no job to process, on the other. The general rule should be to always keep some jobs in the WMS task queue rather than immediately submit them to corresponding CEs. The envisaged solution is to have a control loop for each CE that monitors the number of jobs in the TQ and in the site's batch queue readjusting the queue size when necessary.

Scenario 3: Job Failures

Job failures can be divided into two categories: the one where the failure is caused by an application specific problem and the other where it is due to a problem in the Grid middleware. The first category includes invalid job descriptions, application

software ”bugs” or invalid input data. The cause related to the middleware may be for example some unresolved library dependencies that lead to systematic failures on some jobs. Indeed a job expresses its requirements in a specific JDL file, but there is no fine-grained manner to express precise library dependencies. Therefore a job might be scheduled to run on a WN that does not satisfy the actual job library requirements. Identifying the exact cause of a job failure requires extensive expertise and debugging skills. Furthermore, coordinated investigation over multiple administrative domains is often needed in Grids. To address this problem without resorting to a costly human intervention, it is possible to collect statistics to identify recurring source of failures. A first practical approach consists in building a self-monitoring subsystem that gathers information relevant to job failures. It can then be queried to decide some adaptations based on gradual information about failures as well as statistics such as job executable against failure rate.

Scenario 4: CE Black Hole

Under certain circumstances a CE might malfunction and start to fail all scheduled jobs for some unknown reason. Since it fails all jobs immediately, it will process its queues very quickly hence becoming a *black hole* in the Grid as it will eat all newly incoming jobs that are matched to its configuration. This scenario is not directly linked to failures observed during the data challenge, but it is a well-known issue in the gLite middleware [6]. The self-healing adaptation in this case involves a control loop that monitors execution time, IO activity using low level operating system probes and results of job execution using the appropriate log. When it observes the black hole pattern – a series of jobs with very short execution time and low disk activity – it will put the CE into a drain mode. Drain mode will be reported back to the ISM (Information Supermarket) and after several minutes, the WMS will no longer submit jobs to it. In this scenario there are multiple options on the concrete loop deployment: one control loop per CE or one *master* control loop that manages all CEs in the infrastructure. The different pros and cons of these approaches are to be further experimented as one of the aims of the SALTY framework is to facilitate and capitalize such experimentations.

The following last scenario is related to the *Desktop Fusion* infrastructure management.

Scenario 5: Dynamic Load Balancing

The infrastructure needs to be adapted according to the number of connected users and applications they are using. Taking into account low level resource usage (memory, CPU, etc.) is necessary as well in order to have a better idea of current load of the system. Thus, when a situation of overload will likely happen, a new *Desktop Fusion* server will be deployed. It will be configured as part as the load-balanced alias, so that it will be completely transparent for users. This could handle the case where more and more users are connecting to Desktop Fusion, or if a Desktop Fusion server goes down for some reasons. This scenario has some similarities with the first one on WMS overload and parts of the loops, from models to basic runtime elements, are expected to be shared.

2.1.3 Experimental setup

The initial experimental setup for evaluation of scenarios described below will be done in the SALTY testing environment at MAAT-G. It is a dedicated environment for testing

and experimenting with the implementation of these scenarios. Part of this environment is a WMS that is made especially for the testing purposes. However, it is important to mention that this WMS is connected to the CEs that are part of the production environment of neuGRID so all the tests should be run carefully, not to put any significant load to the underlying infrastructure. For some of the tests an extra effort will need to be done in order to sufficiently test the usability of the approach and of the implementation. This is described per scenario in the detailed appendix.

Besides the initial proof-of-concept tests, executed in the testing environment, there are further plans to incorporate experiments during some larger data and analysis challenges that are part of the future neuGRID project evaluation. Indeed, the latest Data Challenge 3 will consist in validating and assessing the final neuGRID infrastructure. To do this, thousand of MRI scans will be analyzed using at least three different toolkits. This challenge will require a lot of computing power, and the infrastructure will certainly need to be adapted in order to properly handle it and the Salty solution will certainly be really useful for it.

As for the The Desktop Fusion system is based on the FreeNX implementation (`http://freenx.berlios.de/`). This later contains already a load-balancing functionality that will be enhanced by the SALTY add-ons. Moreover, new Desktop Fusion servers will have to be concretely deployed. All the neuGRID infrastructure will be migrated from Xen to XenServer soon and proper Desktop Fusion virtual machines will be created. This will allow the SALTY add-ons to deploy these later contacting the XenServer system.

2.2 "ESB" Middleware Use Case

The second set of scenarii aims to support continuity of services in an ESB platform. This continuity is preserved by using the SALTY framework on its services registry and orchestration engine.

2.2.1 Context

An *Enterprise Service Bus* (ESB) is a middleware infrastructure that provides message-based services for complex systems integration. According to features it provides, it is generally involved in *Service-Oriented Architectures* (SOA). Petals ESB is an implementation of an ESB based on the *Java Business Integration* (JBI) specification, which uses the WSDL and Web services concepts. It is one of the first SOA ESBs and aims at providing SOA features, such as service composability, reusability, loose coupling, security, autonomy, and adaptability. Another feature of Petals ESB is to be built on a natively distributed architecture. A Petals ESB infrastructure consists in a topology of nodes in which various services are deployed. These nodes communicate with each other within a domain, thanks to a dedicated transport layer using a core exchange model. This specificity allows the Petals ESB architectures to be more efficient in distributed and large scale systems than the state-of-the-art ESB implementations. The service composability is performed in Petals ESB by an orchestration engine able to invoke some services in a structured way.

Self-adaptation of such systems is then a key issue for providing a flexible framework, able to face new kinds of services integration and orchestration within the context of dynamic workflows.

In a SOA ESB context, a common issue identified relates to *service composition* and *orchestration*. In particular, a key challenge consists in invoking services within a dynamic

context, which requires adaptations at runtime. We consider for the time being a number of services deployed within a distributed bus and indexed in a distributed registry. Two scenarios have been developed: the first one deals with adaptability of the distributed registry, while the second one addresses a concrete use case that illustrates workflow adaptation needs in the ESB.

2.2.2 Summary of the scenarii

Scenario 1: Self-Organization of the ESB Distributed Registry

In this scenario, we consider a topology of nodes composing the Petals ESB middleware. We define a node containing the master registry while other ones include slave registries. Several services are deployed on each node. The service registry indexes all the available service endpoints and is dynamically updated according to services (un-)installations occurring in the ESB. This mechanism can be prone to instability when nodes fail and, in particular, if it occurs to the node containing the master registry. The SALTY framework can address this weakness by adding an upper monitoring and control layer, based on *Service Component Architecture* (SCA) sensors/effectors, allowing to know the master registry is available in the current topology of nodes and if not, triggering a recovery process to rebuild a consistent distributed registry. Different solutions can be investigated in order to perform this task, either by rebuilding the whole topology of registries and switching a slave registry to a master one, or by selecting a predefined registry to become the new master one. The implementation of this scenario will involve monitoring nodes through a kind of "Heartbeat" pattern, and some appropriate actions to clean the crashed ESB node and redeploy every components of the ESB software stack.

Scenario 2: Self-Adaptation of a Crisis Management Workflow

In this use case, we are interested in workflows, which are performed within an ESB and can be reconfigured at runtime. In such ESBs, a workflow implementation typically consists in an orchestration of services and a well-known standard for such orchestrations is WS-BPEL[1]. It consists in defining activities providing *control structure*, *invocation* and *receive mechanisms*, *correlation* and *compensation features* able to build an execution graph or workflow. This workflow supports several services for working together. They are called *partners* and are defined thanks to their WSDL interfaces. In our context, an adaptation consists in updating an orchestration by changing parts of its execution graph or its partners. Indeed, we believe that reflective mechanisms can help in facing new orchestration issues coming from the complexity increase of business processes, becoming longer in time thanks to asynchronous mechanisms and being involved in more and more dynamic contexts. We build on a legacy use case in order to illustrate the key issues of workflow adaptations. It consists in a crisis management for local authorities, which have to plan for chemical, biological, radiological, and nuclear accident (so called CBRN crisis). This issue, identified in the former ANR SEMEUSE project had some results about dynamic workflows implementing thanks to late binding and semantic match-making extended activities[2]. In this project, we therefore progress on this scenario by addressing the reconfiguration of the workflow itself at runtime, based on non-functional and functional retrieved information.

[1] WS-BPEL: `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`
[2] ANR SEMEUSE Project: `http://www.semeuse.org`

In this scenario, two adaptations are therefore envisaged. The first one consists in taking into account the dynamic aspect of a crisis, faced by the dynamic features of the workflow. Some changes can be reported during a crisis, like an explosion for instance. At the feedback control loop level, workflow reconfiguration can be planned in order to add a branch of execution to manage this explosion. A second adaptation takes into account firemen—reflected as services—functional information (*e.g.*, temperature, heart rate) and corresponds to services invocation reconfiguration. This particular information allows a feedback control loop to detect that firemen are not available anymore and must be evacuated.

2.2.3 Experimental setup

For the first scenario, the experimental setup will be initially done on dedicated testing servers at EBM Websourcing. It will consist in using a set of virtual servers in which Petals nodes will be installed. JBI components will be used upon these Petals nodes, to simulate a realistic middleware environment with SOAP components to provide external Web services to the bus and SCA components to provide execution environment for SCA services with sensors and effectors. For the second scenario, it will consist in using a set of virtual servers in which Petals node(s) and WSDM monitoring node(s) will be installed.

2.3 Geo-tracking Use Case

2.3.1 Context

After the military for guidance and unit follow-up, geotracking popularized itself in several domains such as transportation and logistics. Recently, it is more and more adopted in the context of smartphones where numerous applications like route follow-up but also surrounding services (restaurants, hotels, ...) recommendation become widely available. Applications of geotracking currently explode in many areas: logistic, transportation, security, road traffic control, environmental tax collection on vehicles, etc.

In a nutshell, geotracking uses positioning information of mobile (persons, vehicle, ...) to follow them up in time and space so to use this information for application-specific purposes. Geotracking involves at least two entities: a positioning device and a tracking system. Positioning devices use different techniques to locate themselves. Under the global positioning system (GPS), devices triangulate their positions using signals from satellites. Mobile phones can be located from the geographic locus of the cell to which antenna it is currently connected. Finally, similar techniques can be used to locate WiFi cards from their wireless access points.

Positioning devices transmit positions to a tracking system. Most of the time, mobile phone networks are used for this purpose, but alternatives exist (*e.g.* satellite networks or WiFi). The tracking system is responsible for monitoring the position of mobiles (to which positioning devices are attached) and to trigger reactions when given conditions are met. Typical conditions are: being near, approaching or moving away from some point, approaching or moving away from another mobile, crossing a frontier (in the general sense), ...

Tracking systems can be directly embedded into end-user applications, such as a truck follow-up system, or can use dedicated platforms that correlate position information from several devices to trigger events sent to end-user applications. These platforms act as complex event processing (CEP) systems, but dedicated to geotracking. GeoHub is De-

veryware geotracking platform, which not only correlates positions but also abstract end-user applications from the specifics of positioning devices and positioning techniques.

Geotracking faces two difficulties that will be addressed within the SALTY project. First, sending positions through a mobile phone network incurs a per message cost for customers which need to be minimized while keeping up with application requirements. Linked to this, and sometimes crucial, sending positions also requires energy from batteries which governs the autonomy of the device and so must also be minimized. Second, end-users lack the technical knowledge needed to configure the parameters of the geotracking, like the frequency of position reporting from devices, to catch up with application requirements, like notifying the arrival of a truck at a given location fifteen minutes in advance with a two-minute tolerance. Other points of interest concern fault-tolerance, *i.e.* coping with device malfunctions, and the overall workload of the geotracking platform which can't sustain more than a fixed number of position sendings from all of the connected devices to match its quality of service objectives in event processing (*e.g.* 50,000 positions per minute).

The cost minimization issue will be addressed by dynamically adapting the position reporting, first by modifying its frequency but also by switching back and forth from time-triggered to location-triggered geotracking when possible. As dynamic adaptation is the focus of the SALTY project, complementary adaptation *scenarii* will be provided by the applications themselves. The geotracking configuration, *i.e.* defining events to be notified to applications and the type of adaptations needed at run-time, will be addressed by developing an intelligent interactive configuration system.

2.3.2 Summary of the scenarii

The geotracking use cases is developed into four scenarii:

Scenario 1: Long distance truck tracking

Long distance truck tracking is concerned with following up trucks which deliver goods from warehouses to warehouses in a complex network of logistic bases. This scenario is typical either for general in transportation and large-scale distribution. Three different geotracking objectives are considered:

1. **Notification of arrival at intermediate destinations**, where it is required that trucks arriving at a warehouse notify their arrival enough in advance to let the warehouse coordinator allocate them a port so to optimize port usage and the waiting time of trucks. After notification of their arrival, trucks approaching warehouses are still closely tracked in order to take corrective actions if it appears that they will be too late or way in advance.
2. **Imposed corridor**, where trucks are forced to stay within a corridor around their route to make sure they don't deviate much of this route. When trucks deviate from their corridor, corrective actions are taken, first to inquire drivers for the reason of the deviation, and if justified, replanning the rest of their route. The system will get inputs from traffic control and weather services so that it may confirm or detect itself conditions justifying a replanning of the route.
3. **Waypoint notification**, where the passage nearby some predefined points must be notified. As corridor enforcement deals with deviation from the planned route, waypoint notifications are used to keep track of the progress of the

truck, for example by displaying passed waypoints along with their passage time on a map.

Adaptations are sought to minimize the probability to miss a notification, the power consumption and the cost of data sendings through the GSM network. In this scenario, they include:

1. frequencies of positioning devices, to increase and decrease them so that low frequencies are used when the truck is far from all of its points of interests, but high when approaching these;
2. type of geotracking, either time-based, where positioning devices push data at planed instant, or location-based, where they push data when passing some point;
3. positioning devices themselves, to use either the GPS or the mobile phone of the truck driver when the GPS is malfunctioning;
4. sleep mode, to manage the power consumption of the devices.

Scenario 2: Short distance truck tracking

The short distance truck tracking scenario deals with fast delivery parcel services where delivery men visit customers either to deliver or pick up parcels. Geotracking can help to shorten the time windows imposed to customer for delivery by following more closely the progress of trucks in order to notify customers when the delivery will be late, and replan the route to deliver most customers on time even by skipping some when necessary. The goal is to maximize the number of customers delivered on time, and therefore their satisfaction, even at the expense of a very late delivery to some of them. This scenario involves the following geotracking objectives:

1. **Notification of late arrival**, where trucks must deliver within an time window (with some tolerance) several destinations. When the truck positions and its route show that the delivery of a customer cannot be on time, a notification must be sent to the customer no later than a certain notification delay before the end of his delivery time frame. If the delivery is forecasted to be late, a maximum delay is set, such that when the customer cannot be delivered within this delay, it will be rescheduled later in the day, using route replanning to get a new round for the delivery man.
2. **Imposed corridor**, where trucks are forced to stay inside a certain corridor around their route, otherwise a notification must be sent to the route coordinator. If a route replanning is required, a new corridor will be imposed according to the new route. This objective is essentially the same as in the long distance truck tracking use case, so it will not be detailed again here.

Besides the adaptations recalled from the first scenario, this scenario includes a novel one, the replanning of the round of trucks when clients must be skipped. Such an adaptation distinguishes itself from previous ones by possibly requiring some human intervention to decide upon clients to be skipped and replanned in order to take into account business objectives, like giving priority to the best clients, etc.

Scenario 3: GeoHub QoS enforcement

Given the large-scale nature of the intended system, this scenario looks forward to

adapt of the overall workload of the GeoHub given its current performance. As the delay between position receptions on the GeoHub and the notification of events to applications must be kept under a limit defined by the quality of service offered to customers, massive adaptations of all positioning device frequencies may be required when this delay becomes too large. This scenario will show the capability of the SALTY architecture to cope with large-scale adaptations of distributed systems. The scenario involves two objectives:

1. **Global GeoHub workload management**, where a limit on the overall frequency of position sendings to the GeoHub must be maintained under a certain limit corresponding to Deveryware's QoS objectives. When the workload exceeds this limit, all of the connected positioning devices will be required to lower their frequency, to get a decrease in the overall workload of the GeoHub.
2. **Local frequency limits management**, where each positioning device and their autonomic managers will observe an upper bound on the frequency of its position sendings. Such limits will be adapted at run-time according to the overall workload of the GeoHub and the relative importance of the current geotracking objectives currently driving the use of this device.

Adaptations will mainly concern the frequencies of positioning devices. For the first objective, the adaptation will itself be large-scale, targeting all of the currently connected positioning devices. For the second objective, a more local adaptation is sought, but still requiring care, as heuristics may be used that could potentially develop into a large-scale exploration of the currently connected devices to find one that has available resources to share.

Scenario 4: Decision-making Modeling at Design-time

This scenario is of a different nature compared to the previous ones. One of the goals of the SALTY project is to build an interactive tool, the decision-making modeling at design-time tool, to help non-specialist end-users in eliciting their business objectives and criteria for adaptation for their geotracking applications. As such a tool cannot be fully general, the geotracking use case will provide us with a first context for design exploration. Generality of the tool will be sought through the use of an adaptable database for parameters such as dictionaries, model-driven interaction patterns (question/answer, menu, etc.), etc.

2.3.3 Experimental setup

Testing and experimenting applications that have at least part of their behavior bound to real time is inherently difficult. In the case of truck tracking, the real time behavior comes from the need for real positions sometimes correlated with real geographical positions and artifacts, such as roads and their speed limits, warehouse positions, waypoints, etc.

Hence, it will not suffice to merely generate input data and run an application on a standard computer. The experiment will have to take place in the real time frame, positions and transit times being realistic to some extent. On the other hand, as scalability to large numbers of geotracked vehicles is a key issue in the SALTY project, it is not realistic to get thousands of real trucks going on the road for the experiments; they will require simulated vehicles. Simulated vehicles are pieces of software executing on a computer and sending positions and receiving commands from the GeoHub, as real vehicles do.

The GeoHub, the application and the adaptation layer used in the experimentation will be the real ones[3]. But trucks and their positioning devices will be simulated as threads on stock computers, executing simulated travels. The idea is to get a planned route, using planning services such as ViaMichelin, and then plan a per truck simulation scenario to be run around that route. Each truck simulation will exhibit required notifications, and therefore adaptation *scenarii*. In some cases, incidents, like device malfunctions, traffic jams, etc. will be injected into the *scenarii* to trigger the corresponding adaptations.

As a large number of such truck simulation *scenarii* will be required, they will be generated automatically from the following sub-*scenarii* and their variants. A machine-readable description of these will be constructed, and fed into a simulation *scenarii* generator that will give an executable *scenario* in another machine-readable format used by specifically developed truck and positioning device simulation programs. To our knowledge, no such simulation platform exists to date. It will be a contribution of the SALTY project that could be reused by Deveryware in the future to test other enhancements of their platform.

[3] A copy, for the GeoHub, to avoid service denial to actual customers of Deveryware.

CHAPTER 3

Features

In order to provide key success indicators, the SALTY framework is developed following a feature-driven approach. This approach consists in an iterative and incremental development process, specific to software. It is usually described as an agile method, focusing on a client-valued functionality (feature) perspective. This process is decomposed into five activities: overall model development, feature list building, planning by feature, design by feature, build by feature. The first three activities are usually sequential, whereas the final two activities are iterative and should iterate on features.

In our context, such a feature may range from a functionality of the autonomic framework to some characteristics on how it is built or some resulting properties of using the framework itself. We then first determined the set of features based on the study and classification of the use cases. As the framework architecture will be sketched in the following months, we will determine the success indicators on each feature together with some priorities. This will enable the classic stages of planning, design and build by feature.

In the next paragraphs, the features are organized into four categories, each addressing a particular point of the project:

I. Large-Scale environment,

II. Self-adaptation,

III. Salty Framework building and architecture,

IV. Model-driven engineering in the construction of very large self-adaptive systems.

Inside each category, we define sub-groups of features focusing on main issues to be tackled. For each feature, we briefly present how use cases are related to it. In the description, UC1 will correspond to the Middleware use case. To precise the platform involved by the feature demonstration, we note UC1 GRID to reference the scenarii that are related to the gLite middleware or the desktop fusion bus, and UC1 ESB denotes the scenarii related to the Petals Enterprise Service Bus (ESB). Finally, UC2 will correspond to the Geo-tracking use case.

It must be noted that the list of features presented below is maintained on the internal collaborative web site of the project.

3.1 F.I. Tackling Very-Large-Scale Environments

The following features outline properties induced by self-adaptation of very large systems. We have identified four main dimensions to be dealt with:

A. the intrinsic distribution of these systems and the necessary distribution of the managing infrastructures themselves,

B. the large number and the diversity of managed entities,

C. the large number and the diversity of managing entities,

D. the confidentiality of the managed system.

3.1.1 F.I.A. Supporting the distribution of the managed system and the managing infrastructure

This feature decomposes into the following ones:

FI.A.1. Supporting the distribution of the *managed system*.

FI.A.2. Supporting the distribution of the *managing infrastructure*.

Description

Supporting distribution of the *managed system* includes, for example, the delays in the monitoring of remote services, the need to integrate monitoring data from different remote sites, the time-stamping of all monitoring data to correlate them prior to their use and the coordination of adaptation decisions implying modifications at different remote sites.
Supporting distribution of the *managing infrastructure* includes the need to synchronize distributed adaptations without impacting the performance of the controlled system, and the need to supervise the adaptation with appropriate means (distributed synchronization, multi-agents system approaches, approaches from intelligent and collective robotics to the planning of adaptations).

Link with use cases

FI.A.1. & FI.A.2. UC1 GRID part involves a distributed middleware (1) and all autonomic elements must be coordinated in a distributed way (2).

FI.A.1. & FI.A.2. UC1 ESB part is itself widely distributed (1) and needs to be controlled at node level, that is to say, sensors, effectors and MAPE loops must be disseminated over the ESB topology (2). Thus, UC1 ESB should use scalable architecture as peer-to-peer for sensors.

FI.A.1. & FI.A.2. UC2 is involved as the geoHub infrastructure is a large distributed event-based system (1) and all autonomic elements must be deployed and coordinated appropriately on this infrastructure.

3.1.2 F.I.B. Supporting the large number and the diversity of managed entities

This feature set addresses the following dimensions:

F.I.B.1. Large Amount of data stored, accessed and manipulated.

F.I.B.2. Complexity of connections and interdependencies.

F.I.B.3. Number of computational elements.

F.I.B.4. Number of system purposes.

F.I.B.5. Number of (overlapping) policy domains and enforceable mechanisms.

Description

Large scale systems are known as systems of unprecedented scale in some of the previous dimensions. These systems are then necessarily decentralized in a variety of ways and constructed from heterogeneous parts. This complexity imposes the need for evolving continuously with conflicting needs. In the use cases, we address each of these dimensions to validate the adequacy of the Salty framework when applied on such systems.

Link with use cases

F.I.B.1. & F.I.B.3. UC1 GRID part is concerned by these features, as the scenarii will be applied in the context of the Alzheimer's disease analysis pipeline, which involves very large set of data and is deployed on a neuGRID infrastructure involving a large number of jobs.

F.I.B.2. UC1 GRID part deals with the complexity of connections and interdependencies between the different elements of the GRID. Some of these interdependencies are "implicit" and control mechanisms should be applied to evaluate their interactions.

F.I.B.4. UC1 ESB part addresses this feature as the case study corresponds to a Crisis Management System where the number of alternatives and failures to explore is a priori unknown. Adaptation to the crisis context should support the need for unanticipated dynamic adaptation.

F.I.B.5. All the use cases are potentially concerned by this feature as soon as MAPE loops can interfere in the application itself.

3.1.3 F.I.C. Supporting the large number and the diversity of managing entities

This feature decomposes into the following ones:

F.I.C.1. Large number of adaptable entities.

F.I.C.2. Large number of loops on the same entity or group of entities.

F.I.C.3. Large number of events triggered through loops.

F.I.C.4. Important variability in the controlled system.

Description

Large-scale means that the architecture and the algorithms will need to scale. The control system, made of a large number of MAPE loops over large number of entities, triggering large number of events, must coordinate themselves and implement system-wide adaptations. Peer-to-peer coordination algorithms as well as multi-agents inspired coordination algorithms, are intended to address this issue for the coordination of MAPE loops on a large scale. There is thus a large number of possible architectural patterns of organization between the different involved software entities.

Link with use cases

F.I.C.1. & F.I.C.2. & F.I.C.3. UC2 adaptations have to do with these features, provided that the GeoHub and the positioning devices are viewed as a complex event processing system and probes respectively. Controlling the workload of the GeoHub implies controlling the frequencies of all the positioning devices sending data.

F.I.C.2. & F.I.C.3. & F.I.C.4. UC1 GRID part is concerned by these features, as coupling scenarios lead to several complex loops to be coordinated on the same middleware entities, while triggering a large number of events. Different architectures of the control system are also envisaged.

F.I.C.4. UC1 ESB needs to implement scalable algorithms and frameworks in order to manage system-wide adaptations. It should deal with the variability of the architecture. It is conceivable within the context of a federated architecture with several domains having their proper configurations, service policies and adaptation rules.

3.1.4 F.I.D. Preserving the level of confidentiality of the managed system

Description

Much of the data manipulated in the SALTY use cases have privacy properties. In the Salty project our intent is to support risk analysis, through which targets and vulnerabilities in the system architecture are identified, and specific security requirements are derived from general security objectives (e.g. secrecy of information).

Link with use cases

F.I.D. UC1 GRID part should ensure that access WMS and CE runtime information are restricted to authorized systems.

F.I.D. In UC1 ESB part, security must be managed as one of the main ESB features.

F.I.D. In UC2, positions can be anonymized before being sent to MAPE loops. In some cases, they can be encrypted (programmable devices). Isomorphic transformations can also be applied to make impossible their relation to a precise geographic location for intruders.

3.2 F.II. Supporting the Adaptation of Complex Systems-of-Systems

The following features concerns self-adaptation. According to self-adaptive system classification, these features specifically focus on the dimensions ”Change”, ”Mechanisms” and ”Effects”, as described in section 1.3.
The features are decomposed according to the three following dimensions:

A. Reflecting feedback control loops as first class entities,

B. Supporting the monitoring of heterogeneous and complex data and their quality attributes,

C. Making decisions over complex situations involving multi-criteria objectives.

3.2.1 F.II.A. Reflecting feedback control loops as first class entities

This feature decomposes into the following ones:

F.II.A.1. Being able to coordinate several loops at different levels and scopes (from local to large-scale) making decisions over the same controlled entities in order to produce coherent adaptations.

F.II.A.2. Making the control conscious of its own quality and self-adaptive to achieve stability and efficiency.

F.II.A.3. Attacking control situations with different temporal constraints from tight timing constraints to best effort adaptations.

Description

Given the dynamicity and complexity of large-scale systems of systems, their control loops cannot be defined in advance to implement fixed policies, but rather adapted at run-time. To achieve such adaptations, a reflective representation must be sought to get MAPE loops as first-class entities, which one can reason about, modify, compose, and reuse dynamically. In real-life applications, decisions made by local MAPE loops may affect each others, at all system scope levels from client-server interactions to global constraints. To enforce integrity constraints, such local decisions cannot contradict each others, so the loops must coordinate themselves to achieve collaborative decisions. To deal with a very large set of MAPE loops, we will study coordinated loops based on an hybrid locally hierarchical and globally peer-to-peer architecture, instead of purely hierarchical (which would not scale well). Autonomic managers are components with specific needs in terms of initialization, and meta-control. This can be put under a specific workflow to manage their life-cycles. Autonomic manager, as components, can also be adapted. The architecture shall provide for this by adding autonomic managers to them. The system being distributed and large-scale, MAPE loops need not be always implemented as a unique module but shall take into account the latencies and time requirements to deploy itself in such a way to compute good adaptation policies while satisfying the timing constraints of the control process, (maximum time between triggering conditions and the corresponding execution of the required adaptation).

Link with use cases

F.II.A.3. UC1 GRID is based on best effort adaptation, when UC2 should deal with tight timing constraints.

F.II.A.1. & F.II.A.2. All use cases are concerned by these two features as soon as they mix several adaptations from several scenarii.

3.2.2 F.II.B. Supporting the monitoring of heterogeneous and complex data and their quality attributes

F.II.B.1. Handling explicitly the quality of information in the configuration of the monitoring.

F.II.B.2. Unifying the diversity of probe locations, acquisitions, data types...

Description

MAPE loops will impose stringent requirements on monitoring data, such as being delivered on time (age) and to be coherent (reporting the same state of the system) when several data are used jointly for a decision. State-of-the-Art monitoring frameworks allow for the use of high level QoI-based requests from which configurations for probes are automatically derived. These probes will cover various forms of data types, which will be collected using different protocols from heterogeneous locations (Internet, embedded devices, business processes...). (Includes inputs from the SEMEUSE monitoring, the work on SPACES by INRIA/ADAM, and the work of P8 on smart sensors and fuzzy logic.)

Link with use cases

F.II.B.1. & F.II.B.2. In some sense, most of UC2 adaptations have to do with this feature, provided that the GeoHub and the positioning devices are viewed as a complex event processing system and probes respectively.

F.II.B.1. & F.II.B.2. UC1 will have to deal with monitoring and monitoring QoS because its adaptation mechanisms rest on monitoring probes and would avoid mistakes coming from reports integrity issues.

3.2.3 F.II.C. Making decisions over complex situations involving multi-criteria objectives

F.II.C.1. Handling different types of decision-making problems, from non-sequential to sequential.

F.II.C.2. Tackling controlled entities which dynamics is known either in advance or discovered at run-time.

F.II.C.3. Implementing decision policies coping with multiple criteria of different natures that may contradict each others for given control objectives.

Description

The way decisions made over time influence each others depends upon their impact on the state of the controlled system. In some cases, decisions made at some point change the impact of future decisions. Such problems are tackled in the framework of sequential decision making. Furthermore, most of the time, the precise characteristics of the controlled system cannot be known prior to run-time, because they depend directly upon the deployment context. In these cases, policies cannot be computed statically, but rather dynamically. Most large-scale systems are multi-faceted and adaptation decisions may have contradictory impacts on these different facets. In order to choose upon conflicting decisions, some models will reflect the preferences of the users regarding the conflicting facets, dealing with them as multi-criteria decision making.

Link with use cases

F.II.C.1. UC1 GRID aims to demonstrate the stability of the MAPE Loops.

F.II.C.1. & F.II.C.2.& F.II.C.3. UC2 adaptations deals with multi-criteria policies according to optimization of exchanges number and precision of information.

F.II.C.2. UC1 ESB part on workflow adaptations deals with context evolution that will be dynamically discovered and will imply dynamic adaptations of control entities behavior.

3.2.4 F.II.D. Executing reliable reconfigurations across distributed entities

F.II.D.1. Being able to observe the state of the adaptation process taking place in the control loop.

F.II.D.2. Specifying qualities of service on the adaptation process.

Description

This feature proposes to evaluate the correction and the robustness of the adaptations by analysis of the results. This supplements feature F.II.A. It also suggests authorizing corrections of adaptations. The adaptations themselves could thus be monitored to check their adequacy and to possibly enhance the MAPE-Loop and react by a new adaptation of the system. In addition, adaptation processes may need to be executed with different qualities, from best-effort to transactional, if possible.

Link with use cases

F.II.D.1. & F.II.D.2. UC1 GRID part will experiment adding checking mechanisms to evaluate impacts on MAPE loops integrating such control.

3.3 F.III. Building a Versatile Feedback Control Loop Framework

The features presented here specify paradigms of development that will drive the construction of the Framework itself.
We focus on the following features:

A. component based systems development to produce control systems over service-oriented systems,

B. reusing and sharing the framework artifacts across different domain-specific scenarios by targeting a characterization of the context of use[1].

3.3.1 F.III.A. Adopting SCA as a uniform paradigm to control SOA systems

Description

The control system will be implemented using SCA components, while the controlled system architecture will range from non-SOA (with SCA touch-points for the control system), to SOA and SCA.

[1] According to ISO 9241 definition of context of use : " the nature of the users, tasks and physical and social environments in which a product is used", here product consists in framework artifacts.

Link with use cases

F.III.A. UC1 GRID part will use both SOA and non-SOA infrastructure for the controlled system, and SCA components for the control system.

F.III.A. UC1 ESB part will use embedded SCA components to implement a complete SOA framework, with SCA components both on the controlled and control systems.

F.III.A. In the UC2, both the positioning devices and the GeoHub will be considered as non-SOA and remain as is.

3.3.2 F.III.B. Reusing and sharing the framework artifacts across different domain-specific scenarios

Reuse and sharing are intended to be conducted on:

F.III.B.1. Models

F.III.B.2. Components

F.III.B.3. Loops

Description

The architecture must not be tailored to the use cases, but shall apply to various large-scale distributed autonomic applications.
1) The same model of MAPE loop can be used for several use cases, when different implementations of sensors should be used at deployment time.
2 and 3) In the analysis part of the MAPE loop, the architecture shall allow for the exchange of decision-making techniques by simply exchanging the implementations. The demonstration should come from the ability to interchange a sequential-decision making process with a fuzzy decision-making process as transparently as possible, ideally without changing the interfaces.

Link with use cases

F.III.B.1. & F.III.B.2. & F.III.B.3. Same basic models and control loops should be shared between several scenarii. Components for monitoring, analysis and planning should also be shared and reuse among scenarii. Forms and quantities of shared/reused elements are to be determined as the project progresses.

3.4 F.IV. Designing and Involving Models Continuously

The SALTY project aims at following a model-driven development approach for self-adaptive systems. We thus have the will to identify the steps where the development directed by the models is applicable, as well as the limitations encountered. We will tackle this issue to two points of view:

A. methodology support for the model-driven development of self-adapting systems,

B. consistency management of business requirements all along the lifecycle of the system.

3.4.1 F.IV.A. Adopting a model-driven methodology for the engineering of SALTY

We focus on the following features:

F.IV.A.1. Determining where design-time MDE techniques can be used and applying them.

F.IV.A.2. Defining a representation of the models at runtime.

F.IV.A.3. Providing guidelines in the usage of MDE within the SALTY framework.

Description

MDE approach means a model representation, accessible or not at run-time and linked together and with the code through model transformations amenable to round-trip engineering. In the MAPE loop, models for the analysis can be seen at two levels. A first level concerns only the component architecture at its boundaries (interfaces, membranes, ...) while the second concerns the decision-making processes per se. The latter are sometimes of a more mathematical nature, and therefore way less amenable to current MDE approaches and techniques. However, in this case, models can be apprehended as machine-processable representations stored in the knowledge base of the MAPE-k loop. Though the SALTY project will make contributions to these, no commitment towards a full end-to-end MDE approach can be made at this point.

A model of a system is a description of that system and its environment for some certain purpose. This point has to be taken into account so to design the first level especially to deal with some kind of large scale infrastructures, such as ISP networks. In this case, an efficient model may be built using infrastructure discovery technique or the infrastructure model may be hidden within processing rules.

Consequently, the SALTY project aims at evaluating the interests and limits of model-driven development in the context of Self-Adaptive very Large disTributed sYstems. Guidelines will be extrapolated from our experiments. Justifications will be provided when MDE techniques are not used.

Link with use cases

F.IV.A.1. & F.IV.A.2. All use cases will be finally designed through MDE techniques, but the project progress aims at defining these models incrementally by combining top-down and bottom approaches in the realization of use cases.

F.IV.A.2. In UC2, Model-based Q-learning envisaged for the decision-making process will implement a form of model at run-time, though more mathematically oriented and dealing with the state and dynamic of the controlled system.

F.IV.A.3. All use cases will feed the guidelines content.

3.4.2 F.IV.B. Guaranteeing the propagation and the verification of constraints and agreements throughout the life-cycle of the system

Description

In the autonomic part of the architecture, a meta-control layer will enforce the integrity constraints of the controlled and control system when they correspond to existing and explicit constraints or *Service Level Agreements* (SLA).

Link with use cases

F.IV.B. In UC1 GRID, constraints on the control system such as time between WMS reboots will be taken into account.

F.IV.B. In UC1 ESB, a scenario explicitly links the control loop to pre-existing SLA.

F.IV.B. In UC2, the GeoHub overall workload and performance constraints, as well as geotracking components and their autonomic managers will exhibit such constraints.

CHAPTER 4

Perspectives

This document has set the context and necessary definitions concerning the project. It has also described the requirements of the project use cases, as well as the features that have been determined from them. These features have been organized in categories, a description has been provided for each of them together with their coarse-grain relationships with the use cases' scenarios. The use cases are described in more details in the two last appendices of this report.

Regarding both the definitions and the features, they are already available in some specific collaborative web pages on the site of the project (on its private part). They are going to be maintained all along the project to serve as references. Besides all elements of this report serve as references or starting points for several ongoing tasks of the project. In these tasks, the features will facilitate design by assigning priorities to them and defining success criteria at a finer grain as the project progresses.

- In the work package 2, the refinement of the defined features have begun. Some analysis tables have been provided to all leading partners of modeling tasks related to the managed architecture, the monitoring phase, the decision making part and the execution phase. They are going to be filled to precise the modeling needs in each scenario, as well as which scenario is able to demonstrate which feature. This will then enable the partners to determine easily the priority on the features and a fine grain planning on the design and implementation phases.
- All the previous elements will also drive the overall architecture of the SALTY framework, which is going to be defined as a first partial but functional version at month 12 (as an internal deliverable of work package 1).
- In the work package 4, some first small prototypes have already been designed and implemented on some scenarios. Their incremental developments and experiments will continue, driven by features.

Bibliography

[1] An architectural blueprint for autonomic computing. `http://www-01.ibm.com/software/tivoli/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf`, June 2006.

[2] Jesper Andersson, Rogério de Lemos, Sam Malek, and Danny Weyns. Modeling dimensions of self-adaptive software systems. pages 27–47. 2009.

[3] Grady Booch, James Rumbaugh, and Ivar Jacobson. *Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005.

[4] Yuriy Brun, Giovanna M. Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. Engineering self-adaptive systems through feedback loops. pages 48–70, 2009.

[5] Betty H. Cheng, Rogério Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. Software engineering for self-adaptive systems: A research roadmap. pages 1–26, 2009.

[6] A Duarte, P Nyczyk, A Retico, and D Vicinanza. Monitoring the egee/wlcg grid services. *J. Phys.: Conf. Ser.*, 119:052014, 2008.

[7] Ralph E. Johnson. Frameworks = (components + patterns). *Commun. ACM*, 40(10):39–42, 1997.

[8] E. Laure, F. Hemmer, F. Prelz, S. Beco, S. Fisher, M. Livny, L. Guy, M. Barroso, P. Buncic, Peter Z. Kunszt, A. Di Meglio, A. Aimar, A. Edlund, D. Groep, F. Pacini, M. Sgaravatto, and O. Mulmo. Middleware for the next generation grid infrastructure. (EGEE-PUB-2004-002), 2004.

[9] Diane Lingrand, Johan Montagnat, and Tristan Glatard. Modeling user submission strategies on production grids. In *International Symposium on High Performance Distributed Computing(HPDC'09)*, pages 121–130, June 2009.

[10] David C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[11] J. Miller and J. Mukerji. Mda guide version 1.0.1. Technical report, Object Management Group (OMG), 2003.

[12] L. Northrop, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan, and K. Wallnau. Ultra-Large-Scale Systems

- The Software Challenge of the Future. Technical report, Software Engineering Institute, Carnegie Mellon, June 2006.

[13] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[14] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

APPENDIX

A

Self-Adaptive System Classification

In this section, the overview of the classification of self-adaptive systems from the section *Overview of Modelling Dimensions* from [5] is provided. In depth details about these modeling dimensions together with evaluational uses cases (Traffic Jam Monitoring System and Embedded Mobile System) can be found in [2].

Goals. Goals are objectives the system under consideration should achieve. Goals could either be associated with the lifetime of the system or with scenarios that are related to the system. Moreover, goals can either refer to the self-adaptability aspects of the application, or to the middleware or infrastructure that supports that application. Details are provided in table A.1.

Change. Changes are the cause of adaptation. Whenever the system's context changes the system has to decide whether it needs to adapt. We consider context as any information which is computationally accessible and upon which behavioral variations depend. Actors (entities that interact with the system), the environment (the part of the external world with which the system interacts), and the system itself may contribute to the context that may influence the behavior of the application. Actor-dependent, system-dependent, and environment-dependent variations can occur separately, or in any combination. We classify context-dependable changes of a self-adaptive system in terms of the place in which change has occurred, the type and the frequency of the change, and whether it can be anticipated. All these elements are important for identifying how the system should react to change that occurs during run-time. Details are provided in table A.2.

Mechanisms. This set of dimensions captures the system reaction towards change, which means that they are related to the adaptation process itself. The dimensions associated with this group refer to the type of self-adaptation that is expected, the level of autonomy of the self-adaptation, how self-adaptation is controlled, the impact of self-adaptation in terms of space and time, how responsive is self-adaptation, and how self-adaptation reacts to change. Details are provided in table A.3.

Effects. This set of dimensions capture what is the impact of adaptation upon the system, that is, it deals with the effects of adaptation. While mechanisms for adaptation are properties associated with the adaptation, these dimensions are properties associated with system in which the adaptation takes place. The dimensions associated with this group refer to the criticality of the adaptation, how predictable it is, what are the overheads associated with it, and whether the system is resilient in the face of change. Details are provided in table A.4.

Evolution	This dimension captures whether the goals can change within the lifetime of the system. The number of goals may change, and the goals themselves may also change as the system as a whole evolves. Ranges between: • *static* - changes are not expected, • *dynamic* - goals change at run-time, including the number of goals.
Flexibility	This dimension captures whether the goals are flexible in the way they are expressed. It is related to the level of uncertainty associated with the goal specification. • *rigid* - a prescriptive goal described as a "the system shall do this…" • *constrained* - there is a flexibility as long as certain constraints are satisfied, such as, "the system may do this…as long as it does this…" • *unconstrained* - the statement provides flexibility for dealing with uncertainty such as "the system might do this…"
Duration	This dimension is concerned with the validity of a goal throughout the system's lifetime. Range between: • *temporary* - a temporary goal might be valid for a period of time: short, medium or long. • *persistent* - a persistent goal should be valid throughout the system's lifetime.
Multiplicity	This dimension is related to the number of goals associated with the self-adaptability aspects of the system. • *single goal* - only one goal. • *multiple goals* - more than one goal.
Dependency	In the case system has multiple goals, this dimension captures how the goals are related to each other. • *independent* - goals do not affect each other. • *dependent* - when the goals are dependent they can either be: 1. *complementary* with respect to the objectives that should achieve or they can be 2. *conflicting*.

Table A.1: Goals modeling dimension classification

Source	This dimension identifies the origin of the change depending on the scope of the system. • *external* - a change originating outside of the system, • *internal* - a change originating inside of the system. In this case it might be worth to identify more precisely where change has occurred: application, middleware or infrastructure.
Type	This dimension refers to the nature of change. • *functional* - the purpose of the system has changed and services delivered need to reflect this change. • *non-functional* - system performance and reliability need to be improved. • *technological* - both software and hardware aspects that support the delivery of the service. For example the version of the middleware in which the application runs has been upgraded.
Frequency	This dimension is concerned with how often a particular change occurs. Range between: • *rare* • *frequent*
Anticipation	This dimension captures whether change can be predicted ahead of time. Different self-adaptive techniques are necessary depending on the degree of anticipation. • *foreseen* - taken care of • *forseeable* - planned for • *unforeseen* - not planned for

Table A.2: Change modeling dimension classification

Type	This dimension captures whether adaptation is related to the parameters of the system's components or to the structure of the system. • *parametric* - resulting in parameters adjustment of the system's components. • *structural* - structural adaptation could also be seen as compositional, since it depends on how components are integrated. • *combinational* - a combination of above two.
Autonomy	This dimension identifies the degree of outside intervention during adaptation. Range between: • *autonomous* - at run-time there is no influence external to the system guiding how the system should adapt. • *assisted* - a degree of self-adaptability when externally assisted, either by another system or by human participation (which can be considered another system).
Organization	This dimension captures how the performance of the adaptation is organized. • *centralized* - performed by a single component. • *decentralized* - performance is distributed among several components. No single component has a complete control over the system.
Scope	This dimension identifies whether adaptation is localized or involves the entire system. Range between: *local* and *global*.
Duration	This dimension refers to the period of time in which the system is self-adapting, or in other words, how long the adaptation lasts. • *very short* - less than a second. • *short* - seconds to hours. • *medium* - hours to months. • *long* - months to years.
Timeliness	This dimension captures whether the time period for performing self-adaptation can be guaranteed. Range between: *best-effort* and *guaranteed*. For example, in case change occurs quite often, it may be the case that it is impossible to guarantee that adaptation will take place before another change occurs, in these situations best effort should be pursued.
Triggering	This dimension identifies in what way the change that initiates adaptation. Although it is difficult to control how and when change occurs, it is possible to control how and when the adaptation should react to a certain change. • *event-trigger* • *time-trigger*

Table A.3: Mechanisms modeling dimension classification

Criticality	This dimension captures the impact upon the system in case the self-adaptation fails. There are adaptations that harmless in the context of the services provided by the system, while there are adaptations that might involve the loss of life. • *harmless* • *mission-critical* • *safety-critical*
Predictability	This dimension identifies whether the consequences of self-adaptation can be predictable both in value and time. While timeliness is related to the adaptation mechanisms, predictability is associated with system. Range between: *non-deterministic* and *deterministic*.
Overhead	This dimension captures the negative impact of system adaptation upon the system's performance. Range between: *insignificant* and *system-failure* - when the system ceases to be able to deliver its services due to the high-overhead of running the self-adaptation processes (monitoring, analyzer, planning, effecting processes).
Resilience	This dimension is related to the persistence of service delivery that can justifiably be trusted, when facing changes [6]. There are two issues that need to be considered under this dimension: first, it is the ability of the system to provide resilience, and second, it is the ability to justify the provided resilience. Range between: *resilient* and *vulnerable*

Table A.4: Effects modeling dimension classification

https://salty.unice.fr/

ANR SALTY

Self-Adaptive very Large disTributed sYstems

Work Package:	WP1 - Requirements and Architecture
Coordinator:	UNS
Deliverable:	D-1.1 - Appendix B
Title:	**Middleware Scenarios Specification**
Submission date:	2^{nd} August 2010
Project start date:	1^{st} November 2009, duration: 36 months
Revision:	209
Last change:	02.08.2010

Authors

(for this appendix)

Author	**Affiliation**	**Role**
P. Collet	UNS	Lead
D. Manset	MAAT-G	Lead
M. Blay-Fornarino	UNS	Writer
F. Křikava	UNS	Writer
J. Lesbegueries	EBM Petals Link	Writer
R. Mollon	MAAT-G	Writer
J. Montagnat	UNS	Writer
J. Revillard	MAAT-G	Writer
R. Rouvoy	INRIA Lille	Writer

Contents

CHAPTER 1

Introduction

The objective of this appendix is to analyze middleware use cases and in particular to validate self-adaptive SALTY features on Grids and Enterprise Service Bus (ESB).

The Grid is used in the context of the Alzheimer's disease analysis pipeline use case, which is currently deployed on the neuGRID infrastructure. Scenarios for demonstration and validation of the SALTY project are to be determined. The ESB is used in a technical context and in a crisis management use case in order to validate self adaptive features for the bus integrity and especially the use of services deployed on it. This use case is built upon the Petals ESB infrastructure.

The remainder of this appendix is organized as follows. The introduction describes the context. Section 2 describes a family of scenarios related to the self-optimization of the gLite Grid middleware. Another family of scenarios concerns the self-configuration of Desktop Fusion application and is described in section 3. Section 4 details ESB scenarios dealing with self adaptive features.

1.1 Context

1.1.1 Grid and Desktop Fusion Context

The neuGRID European infrastructure aims to support the neuroscience community in carrying out research on neurodegenerative diseases. In neuGRID, the collection of large amounts of imaging data is paired with grid-based computationally intensive data analyses. The infrastructure is developed to run neuro-imaging and data-mining pipelines of algorithms, in particular specializing on Alzheimer's disease[1] research with the analysis of cortical thickness from 3D Magnetic Resonance (MR) brain images. Capitalizing on the databases acquired in the US[2] and Europe[3] respectively, up to 13,000 MR scans of brains should ultimately be archived in the infrastructure, thus constituting the largest ever standardized database in the field. Expected to complete in early 2011, neuGRID will provide neuroscientists and potential pharmaceutical industries with a harmonized framework and powerful distributed environment to seamlessly create, use, combine and validate algorithm pipelines to process acquired data and thus support clinical trials activity.

The neuGRID project is the first project within the neuroscientific community to use the Grid technology. Pipelines manipulated in neuGRID are computationally intensive as they enact a mixture of both short and long running I/O demanding algorithms that are applied over large data sets containing tens of thousands of images. It thus brings underlying Grid resources to their limits and highlights technological bottlenecks which must be addressed through appropriate scheduling optimization, data replication and gridification fine tuning. As an example, the formerly cited cortical thickness pipeline

[1] It is the second most feared disease associated with aging, following cancer, according to the Alzheimer Society of Canada

[2] ADNI Project `http://www.adni-info.org`

[3] EU-ADNI Project `http://www.centroalzheimer.it/E-ADNI_project.htm`

takes approximately 15 hours of CPU time when executed on a regular workstation and applied to only one brain. In the context of population pattern searching, applying the cortical thickness over 13,000 scans would simply be a waste of time with a single PC, bringing it to 22 CPU-years. In the target deployment of the neuGRID project with 4 European sites, each hosting about 20 quad-core CPUs paired with 5TB of effective storage, the execution time of the example case could shrink down to matter of weeks. To enable such a massive amount of data and to adequately service on demand computing power, neuGRID is utilizing a Grid infrastructure based on the gLite middleware [5].

1.1.2 ESB Context

An Enterprise Service Bus (ESB) is a middleware architecture which provides message-based services for complex systems integration. According to features it provides, it is generally involved in Service Oriented Architectures (SOA).

Petals ESB is an implementation of an ESB based on JBI[4] specification that uses WSDLs and Web Services concepts. It is one of the first SOA ESBs and aims at providing SOA features such as service composability, reusability, loose coupling, security, autonomy and adaptability. It is a part of PetalsLink suite for SOA, that also contains Business Activity Monitoring and Governance tools.

Another feature of Petals ESB is to be built on a distributed architecture. A Petals ESB infrastructure consists in a topology of nodes in which various services are deployed. These nodes communicate between each other within a domain. Domains can consist in Petals topologies or other middleware architectures that communicate over Internet. This specificity allows Petals ESB architectures to be more efficient in distributed and large scale systems. Self-adaptation of such systems is then a key issue for providing a step forward framework, able to face new kinds of services integration and orchestration within the context of dynamic workflows.

[4] Java Business Integration.

CHAPTER 2

Scenario 1 - Grid Self-Management

The first scenario is in the context of Grid Computing. We consider a large Grid infrastructure that supports scientific computing, medical image analysis in particular.

2.1 Context

In this section we first introduce the general motivation behind this set of scenarios, the difficulties in nowadays Grid computing, and by doing that we prepare the foundation for further presenting the autonomic computing as a mechanism for tacking these issues. Next we briefly outline the core principles of the gLite Grid middleware in regards to the job management and in the remaining subsection we describe the details of the neuGRID project which is the Grid whose operation motivated the scenarios and in which they are going to be implemented and experimented.

2.1.1 Introduction

Grid infrastructures have become a critical substrate for supporting scientific computations in many different application areas. Over the last decade, world-wide scale Grids (*e.g.* EGEE[1], OSG[2], PRAGMA[3]) leveraging the Internet capabilities have been progressively deployed and exploited in production by large international consortia. They are grounded on new middleware federating the grid resources and administration frameworks and enabling the proper operation of the global system 24/7. Despite all efforts invested both in software development to achieve reliable middleware and in system operations to deliver high quality of service, grids encounter difficulties to implement the promise of ubiquitous, seamless and transparent computing.

The causes are diverse and rather well identified. They notably include i the complexity of middleware stacks, making it extremely difficult to validate code; ii the dependence of the overall infrastructure to many distributed resources (servers, network) which are prone to hardware failures and exogenous interventions; iii the heterogeneity of hardware and software operated, leading to almost infinite combinations of inter-dependencies; iv the uncontrolled reliability of the application codes enacted that sometimes has side-effects on the infrastructure; v the incompatibilities between software components although they were meant to be interoperable; vi the difficulty to identify sources of errors in a distributed, multi-administrative domains environment; vii the challenging scale of the computing problems tackled; etc.

The practice demonstrates that the human administration cost for grids is high, and end-users are not completely shielded from the system heterogeneity and faults. Heavy-

[1] Enabling Grids for E-sciencE, `http://www.eu-egee.org`
[2] Open Science Grid, `http://www.opensciencegrid.org`
[3] Pacific Rim Applications and Grid Middleware Assembly,`http://www.pragma-grid.net`

weight operation procedures are implemented by the grid administrators and users have to explicitly deal with unreliability issues [6].

Acknowledging the fact that middleware can hardly achieve complete reliability in such a challenging context, new operation modes have to be implemented to make grid systems resilient and capable of recovering from unexpected failures. Recently, there has been a lot of effort put into considering alternative paradigms and techniques that are based on principles used by biological system or in control engineering. These approaches, referred to as Autonomic Computing, aim at realizing computing systems and applications managing themselves with minimal or none human intervention [9]. Such systems then provide some self-management properties, mainly *self-configuration*, *self-healing*, *self-optimization*, *self-monitoring* and *self-protection*.

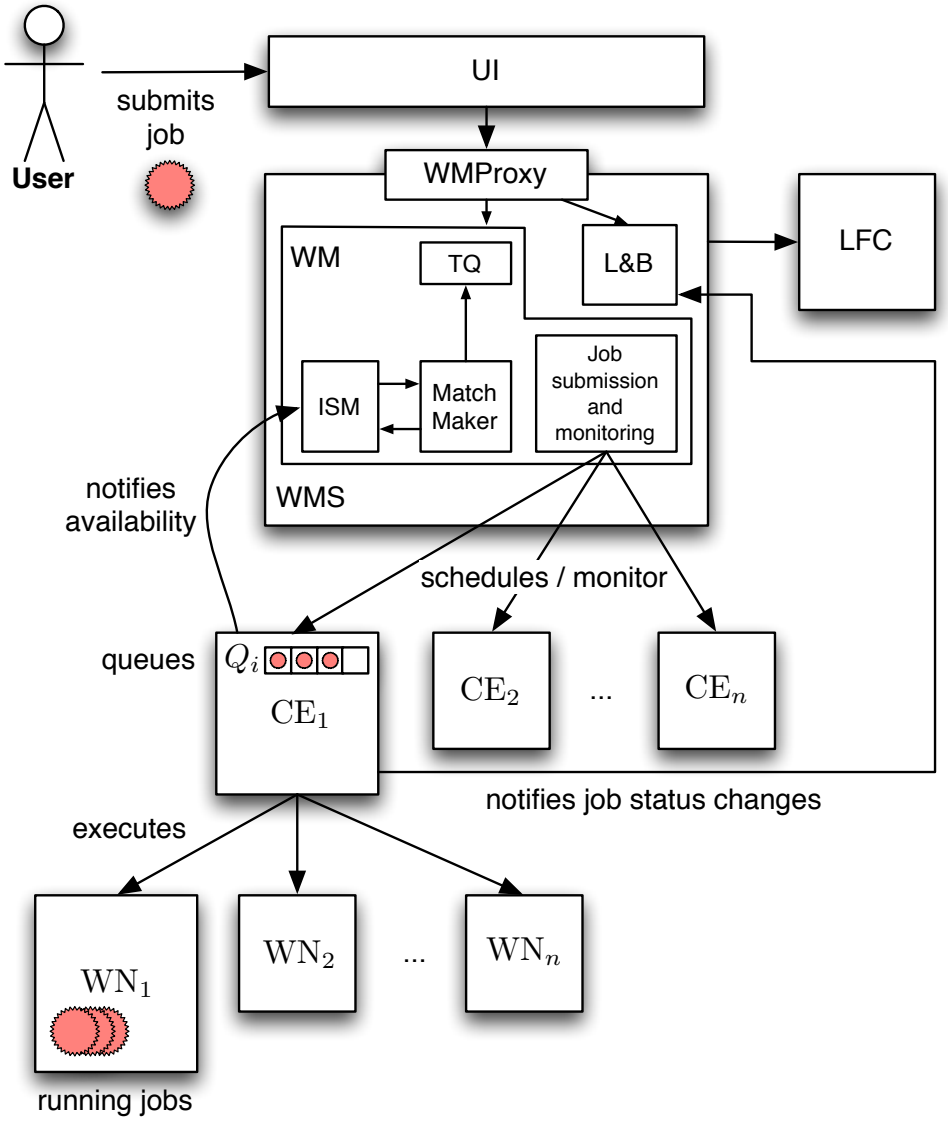
2.1.2 gLite

The gLite middleware has been developed as a part of the European project EGEE which delivers a reliable and dependable European Grid infrastructure for e-Science. gLite is architected as a two-levels batch system that federates resources delivered by multiple computing *sites*. Each site is exposing its *Worker Nodes* computing units (WN) through a *Computing Element* (CE) gateway. A high-level meta-scheduler called the *Workload Management System* (WMS) is used as a front end to multiple CEs.

Grid applications are sliced in smaller computing *jobs*. Each job is described through a *Job Description Language* (JDL) document that describes the executable code to invoke and specifies the specific requirements associated[4]. Jobs are submitted from a client *User Interface* (UI) to the WMS. The WMS is responsible for resources identification and job management across Grid resources, in such a way that jobs are conveniently, efficiently and effectively executed (fig. 2.1). Effectively, the job enters the WMS through a simple web service base interface (WMProxy) and it is passed to the *Workload Manager* (WM) to be queued into a file system-based *Task Queue* (TQ). A matchmaking operation then takes place to identify available and suitable resources. The matchmaking is done by interrogating the *Information Supermarket* (ISM), an internal information cache, to determine the status and availability of computational and storage resources and query the *Logical File Catalogue* (LFC) to find locations of any required input files. Once an appropriate CE has been found the WMS delegates the job processing to the CE batch manager where it is queued until a WN can process it. The job scheduling policy configured in neuGRID's WMS is eager scheduling, where a job is matched against the resources and passed for execution as soon as possible.

A WMS can be configured to use different policies regarding the job scheduling task. On one extreme, it can use an eager scheduling where a job is matched against the resources and passed for execution as soon as possible, where, it will very likely end up in a queue. The other extreme is lazy scheduling, where a job is being held by the WMS until a resource has become available, at which point the resource is matched against the waiting jobs (from TQ) and the one which does fits the best is pushed to that resource for immediate execution. The main difference is that at the match-making level in the former scheduling the job is matched against multiple resources whereas the latter implies matching a single resource against multiple waiting jobs in the WMS. At the queue level, non-matching requests will be retried either periodically (*eager*) or a soon as an available resource will appear (*lazy*) in the ISM.

[4]These are specified by a user and usable according to the local policies specified by the local system administrator.



gLite submission process: A user submits a job using **UI** (User Interface) into **WMS** (Workload Management System), where it is put into a **TQ** (task Queue), before it is matched against available **CEs** (Computing Elements) using information from the **ISM** (Information Supermarket) and **LFC** (Logical File Catalog). Once a suitable **CE** is available a job is submitted into its queue. The actual execution is performed on a **WN** (Worker Node). The job can make use of a **R-GMA** (Relational Grid Monitoring Architecture). All job state transitions are logged by **L&B** (Logging and Bookkeeping service).

Figure 2.1: gLite job submission overview (a logical schema)

.

Similarly, CEs can operate in a push mode, where the job submission is triggered by other services, in particular by the WMS, and in a pull mode, where the job dispatching is initiated by CE itself [1]. In push mode, when a job is sent into CE, it gets accepted only if the CE has an available space in a queue. The job gets then dispatched to a working node matching these constraints. In the pull model, on the other hand, when CE can receive a job (for instance the local queue is empty of it is getting empty) it requests it from a well known WMS.

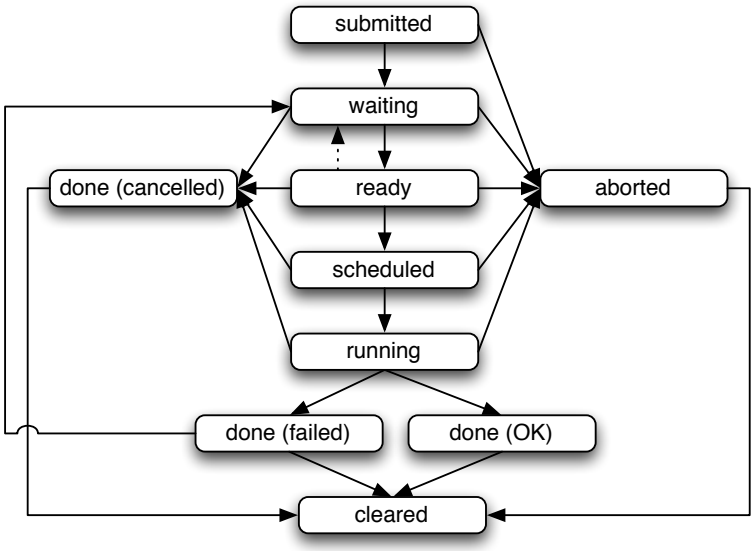


Figure 2.2: Job state machine

When submitted, a job goes through a series of states (as shown in fig. 2.2). The change from one state to another as well as other important events in the job life-cycle, like finding a matching CE, are being tracked by the *Logging and Bookkeeping service* (L&B). These events are being gathered by WMS components as well as CEs. Events are passed to a physically close component of the L&B infrastructure in order to avoid any sort of network problems. These components are responsible for persisting events and delivering them to one of the bookkeeping servers [5]. This server process them and provides a higher level view of the job states (*submitted, running, done, ...*) together with various attributes like the job's JDL[6], matched CE, exit code, etc. It is also possible for the user, instead of querying the L&B service directly, to register itself to be notified on a particular job state changes - for instance when it finishes.

More information about gLite components is available in [1]. For efficient job submission it is necessary to adjust the size of a job queue at the CE level accordingly to the type of jobs that are being submitted into a Grid in order to decrease the time needed for a job to start executing (time needed to move from submitted state to running in the job state machine) and to prevent time-outs with potential jobs resubmissions.

[5] assigned statically to a job upon submission
[6] Job Description Language - a language for describing a job with all its requirements and dependencies

2.1.3 neuGRID Data Challenge

A part of the neuGRID project is a set of validation tests that are run within the infrastructure in order to verify its good performance while meeting user requirements specification. These performance tests are executed in form of *data challenges* in which a very large data set of medical images is analyzed hence putting a lot of stress onto the underlying infrastructure.

During the neuGRID project there are 3 data challenges, two of them have already been run and one is scheduled. The differs in the aim as described below:

Data Challenge 1 testing the infrastructure in general - to make a test of the entire neuGRID gLite middleware stack in an automatic way. It divides the test into five different areas:

- Security services
- Information system services
- Data management services
- Job management services

Data Challenge 2 testing the performance of the entire neuGRID infrastructure in order to determine directions for ongoing developments.

Data Challenge 3 validating and assessing the final neuGRID infrastructure (performance, limitations), in order to comfort technical decisions or to indicate possible alternatives to be chased post projects.

The neuGRID infrastructure (as shown in fig. 2.3) is composed of two levels:

LEVEL 0 represents the "infrastructure ground truth" level with *Grid Core Center* (GCC). It includes Grid central core services for Authentication (MyProxy, CA), Authorization (VOMS), LFC, WMS and central Information System Service (top BDII).

LEVEL 1 represents the Data Archiving and Computing Sites (DACS). In other words, this level encompasses distributed storage (DPM, a Storage Element implementation) and computing (CE/WNs) resources. Next to them stands another Information System Service (site BDII), which is site-specific.

In the rest of this section we will focus on the second data challenge which has been used to serve as a start point for identifying representative scenarios presented in this document.

The data challenge consisted in analyzing the entire dataset of the US-ADNI data using the CIVET pipeline[7] that contains 715 patients with 6'235 scans in MINC[8] format, representing roughly 108 GB of data. Each scan is about 10 to 20 MB and contains between 150 to 250 slices. The experiment was running for less than 2 weeks[9] producing about a 1TB of data and utilizing in the peak 184 cores in parallel.

It was executed as a *parametric* job that was submitted into the gLite middleware in the very same way as presented in the section 2.1.2. A parametric job is a job that allows

[7] CIVET pipeline `http://wiki.bic.mni.mcgill.ca/index.php/CIVET`

[8] Medical Image NetCDF `http://www.nitrc.org/projects/minc/`

[9] just for comparison, it would have taken couple of years to accomplish the same using a single workstation

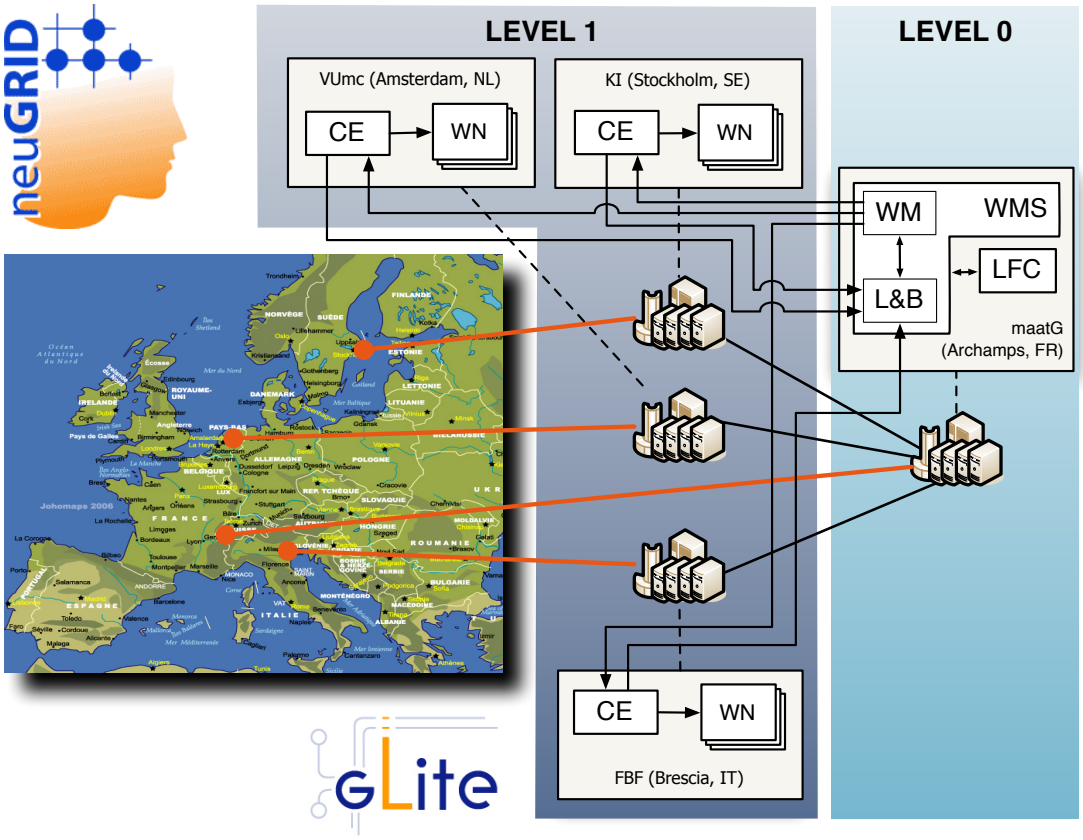


Figure 2.3: neuGRID deployment

to create a bulk of similar jobs that only differ in arguments and submits them as a single job. The WMS then breaks the parametric job into respective number of single jobs and submits them separately into CEs on users behalf, thus significantly reducing the time needed for the jobs submission.

During the data challenges several problems have been observed that required significant intervention from the operating personnel not only resulting in prolonged execution time but more importantly in higher cost. They vary in nature (hardware/middleware/application) and severity.

Representative hardware failures encountered are:

- A power-failure resulting in the whole site (CE) going down, had to be recovered. Submitted jobs were pushed to the other CE where they caused a failure due to an overload (below).

Representative middleware failures encountered are:

- WMS service overload - not able to handle all submitted jobs, had to be manually reconfigured and restarted.
- WMS service crashing - due to memory leak in the middleware, had to be manually restarted and pending jobs rescheduled.

- CE service overload - not able to handle all submitted jobs, had to be manually reconfigured and restarted.
- LFC service overload - not able to handle too many requests from the many services within the Grid, necessitated a workaround handling timeouts to be developed because of LFC not responding.

Representative application failures encountered are:

- Library incompatibilities between CIVET pipeline and WN operating system - very difficult to trace down what is the exact cause, had to be manually rescheduled.
- Bad data - ADNI images not fully quality assessed, manually rescheduled in cases where the part of work flow that could be recovered.
- Problems in the pipeline itself, had to be manually rescheduled.

All impacts of the described issues are quite significant to the normal operation and maintenance of the grid. Consequently, managing such problems through additional autonomic capabilities are likely to bring important benefits on other data challenge runs and on the normal day-to-day operation of the infrastructure.

The presented scenarios are motivated by middleware related issues from the data challenge experiment, but also by recurring issues on the EGEE Grid in which the gLite middleware is also deployed. Consequently, we can define the overall system goal of this middleware to be the following.

System Goal The system shall allow for stable distribution and management of jobs across Grid resource in such a way that applications are efficiently executed.

2.2 Overall Experimental Setup

The initial experimental setup for evaluation of scenarios described below will be done in the SALTY testing environment at MAAT-G. It is a dedicated environment for testing and experimenting with the implementation of these scenarios. Part of this environment is a WMS that is made especially for the testing purposes. However, it is important to mention that this WMS is connected to the CEs that are part of the production environment of neuGRID so all the tests should be run carefully, not to put any significant load to the underlying infrastructure.

For some of the tests an extra effort will need to be done in order to sufficiently test the usability of the approach and of the implementation. These will be discussed in greater details at the appropriate section of each scenario.

Besides the initial proof-of-concept tests, executed in the testing environment, there are further plans to incorporate experiments during some larger data and analysis challenges that are part of the future neuGRID project evaluation. Indeed, as it was describe in the section 2.1.3, the latest Data Challenge 3 will consist in validating and assessing the final neuGRID infrastructure. To do this, thousand of MRI scans will be analyzed using at least tree different toolkits. This challenge will require a lot of computing power, and the infrastructure will certainly need to be adapted in order to properly handle it and the Salty solution will certainly be really useful for it.

2.3 Scenario 1.1 - WMS Overload

2.3.1 Objective

The WMS component is the gateway to the gLite job management system. Most of the requests regarding the job submission, status, etc. involve the WMS and therefore it might get overloaded. The overload is usually caused by receiving more requests that it can handle or due to a software problem in the component itself, *e.g.* a memory leak such as the one encountered during the data challenge.

To deal with this kind of failure, an additional self-healing control loop should be deployed into the infrastructure. This loop interacts with the WMS host's low level operating system probes and periodically monitors CPU and memory utilization done by WMS process. We define two threshold values for the system. When the first one is reached then WMS is being blocked from all new incoming jobs to be submitted until its resource usage either goes below this threshold or till it reaches the second threshold. If that happen the adaptation mechanism will proceed and restart WMS.

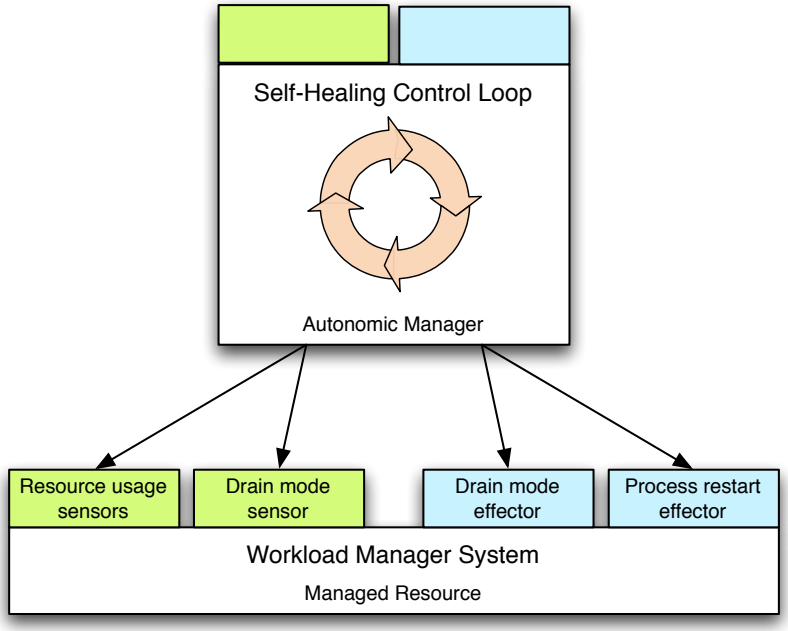


Figure 2.4: WMS overload schema

The reason for establishing two thresholds is to give a chance to WMS to potentially recover from a high load or at least allow it to schedule as many jobs as possible to the computation sites before it is taken down so the final impact on the infrastructure is smaller.

2.3.2 Classification

Below we provide an analysis of the described adaptation in the context of the modeling dimensions from appendix A:

Goals. The self-healing subsystem shall lower the impact caused by the WMS overload.

Evolution: *Static*
In this scenario only a single goal is considered and it does not change within the lifetime of the system.

Flexibility *Rigid*
The goal is prescriptive.

Duration *Persistent*
The goal is valid throughout the system's lifetime.

Multiplicity *Single goal*
In this scenario we consider a system having only one goal.

Dependency *N/A*
In this scenario we consider a system having only one goal.

Change. The adaptation is triggered when resource usage of the WMS process reaches certain thresholds.

Source *Internal*
The change in resource usage is internal to the system and happens in the middleware (WMS is apart of the gLite middleware).

Type *Non-functional*
The change is related to QoS of the job management service.

Frequency *Rare* - normal operation; *Seldom* - during data challenge
During the normal grid operation, the system's resource usage should stay within specified boundaries. During a data challenge the overload is however more likely to occur.

Anticipation *Foreseen*
Although overloads are undesirable, they should be expected hence the overload of WMS (during data challenges in particular) should be considered as foreseen.

Mechanism. In case the system has reached the first (blocking) threshold, the mechanism should prevent the system from accepting any new job submission requests. In case of system reaching the other (restarting threshold), the mechanism should restart the WMS process.

Type *Parametric*
Both mechanism are parametric, as a contrary to structural. The first adaptation changes the WMS behavior through and in the second adaptation, restarting the WMS can be seen as setting a technical parameter (restart) on it.

Autonomy *Autonomous*
No outside intervention.

Organization *Centralized*
The adaptation is performed by a single centralized component.

Scope *Local*
Adaptation only involves changes in WMS component.

Duration *Short*
The amount of time required to reconfigure the system should be short. Reconfiguration impacts the availability of (some of) the system's services.

Timeliness *Best-Effort*
Enabling or disabling drain mode is done externally through a file content which is reread by the system in way that cannot be affected from the outside. The restart action also might last longer than expected. In any case, there need to be guarantee that only one adaptation processes at a time.

Triggering *Event-trigger*
Both adaptations are triggered when resource usage reaches certain value.

Effects. System temporarily stops accepting new jobs or is restart.

Criticality *Mission-critical*
It is mission-critical as we cannot guarantee that the WMS will come back online after its restart. In case of failing the adaptation mechanism that puts it into or out of a drain mode the entire job submission mechanism is compromised. The very applies in case of system restart.

Predictability *Deterministic*
In both cases, there the actions are well determined.

Overhead *Reasonable*
There should be a mechanism to ensure the system does not constantly reconfigure itself. This is realized by ensuring that the adaptation is triggered only if there are significant changes in the monitored data over a pre-specified period of time. However, there is a considerable overhead in terms of wasted resources as during the reconfiguration some of the system's services might become unavailable.

Resilience *Vulnerable*
WMS become temporary unavailable while being restarted and partly unavailable (to new submission requests) while being in the drain mode.

2.3.3 Adaptation

An overload is detected when resource utilization exceeds a certain threshold value (fig. 2.5). We define two threshold values with associated adaptation mechanisms:

1. *blocking* threshold T_0 and
2. *restarting* threshold $T_1, (T_0 > T_1)$.

When the loop detects that T_0 is exceeded, the adaptation mechanism will block all incoming jobs from entering the system. It does that by putting WMProxy into a drain mode that prevents it from accepting any new job submission requests. This should remove a part of the load and therefore increases a chance for WMS to recover[10]. It will also allow the service to process as many already queued jobs from its TQ as it can, before the resource usage reaches the second threshold T_1. At the point when T_1 has been exceeded, the adaptation mechanism will restart the WMS process itself. All the job management services, together with monitoring will ceased for the duration of the service restart t_r. If the system has recovered and its resource usage dropped below the blocking threshold T_0, the WMProxy will again be enabled to accept new job submission requests.

[10] unless the overload is caused by a software defect

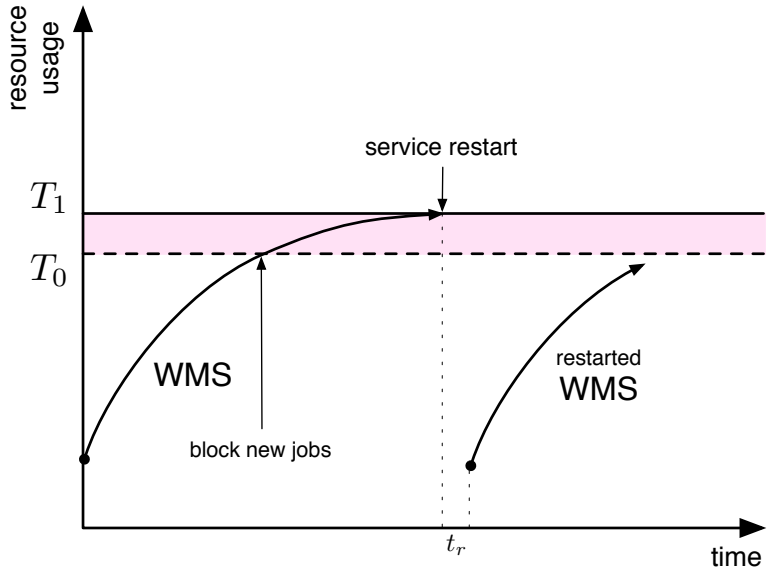


Figure 2.5: WMS overload model

At first, both T_0 and T_1 are empirical, but the next step is to make them to evolve during the system life time so they adapt to the current system context (see extensions 2.3.5). The actual adaptation model is subject for further investigation. It is currently based on automatic threshold setting techniques [3].

The monitoring part of the control loop should also be self-adaptive. Instead of taking the resource usage samples at a constant rate, it should adapt the rate frequency based on the load observed in the system. The higher the load is, the shorter the sampling intervals should be in order to have a very precise information about the system and execute the adaptation policy on time.

The figure 2.6 illustrates the adaptation of sampling rate according to the resource utilization. The concrete model of the monitoring adaptation is also to be improved and simple statistical models are intended to be experimented first [7].

In this section we refer to resource usage as a mean to observe the load of the WMS. There are few metrics that can be used for that. The major one include memory and CPU usage. The loop need to keep historical values of these measurements to build its internal model of resource consumption of the WMS in time for better analysis of the state.

2.3.4 Experimental Setup

The experimental setup will be initially done in the dedicated testing environment at MAAT-G. The subject of the adaptation in this scenario is WMS and the aim is to verify that the feedback control loop correctly reacts upon the overload of the WMS. In the testing environment we cannot submit that many jobs as in the end they will be executed by the production part of the neuGRID. Therefore we will simulate the load by lowering the thresholds. In this way we can simulate overload even with only a few jobs. Although it is not the very same as stressing the WMS to its limits, it should be sufficient to verify the implementation and get some initial feedback.

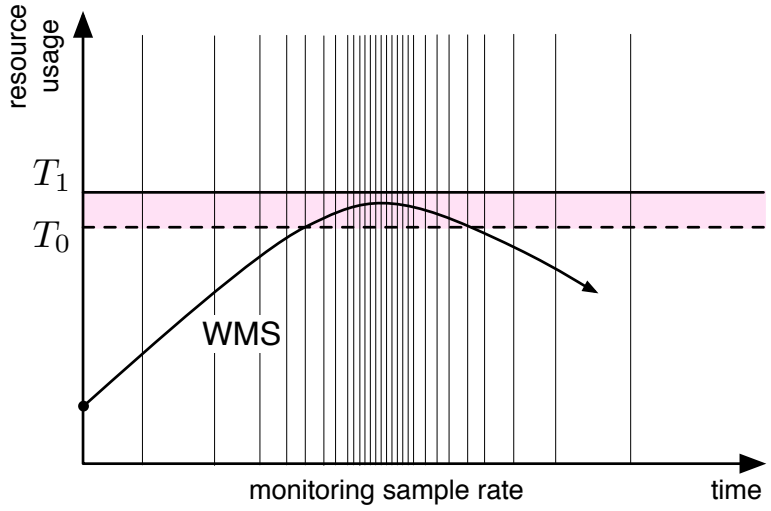


Figure 2.6: WMS resource usage adaptive monitoring

2.3.5 Extensions

We describe here some possible extensions of this first scenario. Each extension leads to additional complexity in different areas of the feedback control loop architecture and the overall behavior.

- Taking into an account the other scenarios, we can aggregate probes among different feedback control loops and compose them into some hierarchy, where one loop can controls others, etc. One concrete example is to take a case of CE Starvation (cf. section 2.4) and add a self-protecting subsystem build on the top that. It will be responsible to disconnect CE starvation loop when it finds out that it puts too much load on the WMS.

- Similarly to the previous point, there can be another loop on the top of the one responsible for the WMS self-healing. This one will on a longer term monitors the effects of adaptations triggered by the other one and it will try to reason about whether the overall system performance is actually improved or not. Based on that it might adjust the other loop properties (e.g. thresholds), etc. Another example can be a loop that monitors the frequency of adaptations of the self-healing loop and puts some restrictions on the rate of those adaptations.

- The WMS system is described as a whole in this scenario. However, it can be decomposed into individual components (i.e. WMProxy, Workload Manager, cf. 2.1). Then one can introduce managed resources to these components, as well as loops that coordinate them on the top. In this case one can monitor for example resource usage of the Workload Manager (WM) and based on that adjust the amount of requests coming from WMProxy.

2.4 Scenario 1.2 - CE Starvation

2.4.1 Objective

During the data challenge run (fig. 2.7), when the CE had disappeared because of the power failure, the WMS correctly detected the situation and rescheduled all jobs to the other site that remained available. However, the sudden schedule of many jobs resulted in an complete overload on the other site that had to be restarted in the end. This could have been fixed by setting a smaller queue size. Nevertheless, this introduces a different but more severe issue. If the site receiving all rescheduled jobs was not overloaded and continued to work and the other site appeared again, it would have no jobs to execute. This would result into the situation when one site is very busy and the other completely idle, being able to only work on newly arrived jobs. Therefore, in this scenario, the objective is to keep all computing elements optimally utilized and prevent them from both extremes: an overload, due to large number of jobs getting scheduled, on one hand and a starvation, with no job to process, on the other.

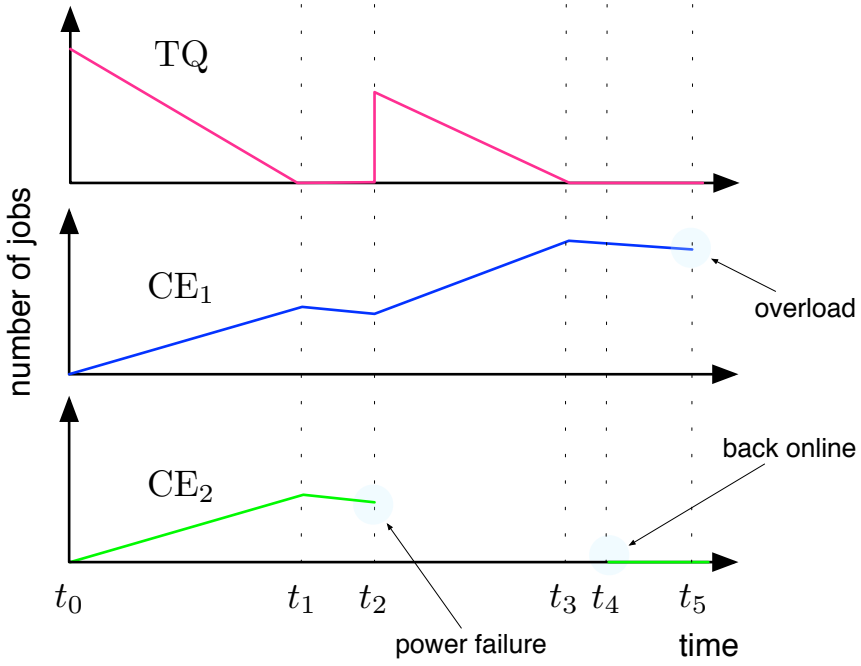


Figure 2.7: CE starvation as demonstrated during the data challenge

The general rule should be to always keep some jobs in the WMS task queue rather than immediately submit them to corresponding CEs. The standard behavior of WMS[11] is that it schedules a job as soon as there is a matching CE resource available i.e. when it has a free slot in its batch queue. So in order to keep jobs in TQ the size of the queue at CE level must be set to reasonably small number according to the context. On the other hand, the number should not be too small, because then when the execution time of jobs is short, the site will be running out of work to do.

[11] WMS that is configured in eager scheduling mode, like are the ones in neuGRID or EGEE.

2.4.2 Classification

Below we provide an analysis of the described adaptation in the context of the modeling dimensions from appendix A:

Goals. The self-optimization subsystem shall allow for better distribution of jobs among the computing sites available in the infrastructure and prevent system from having some sites overloaded and some sites idle.

Evolution: *Static*
In this scenario only a single goal is considered and it does not change within the lifetime of the system.

Flexibility *Rigid*
The goal is prescriptive.

Duration *Persistent*
The goal is valid throughout the system's lifetime.

Multiplicity *Single goal*
In this scenario we consider a system having only one goal.

Dependency *N/A*
In this scenario we consider a system having only one goal.

Change. The adaptation is triggered by the number of jobs in the CE job queue.

Source *Internal*
We consider the cause of the adaptation to be the fluctuation of the number of jobs in a job queue at the CE level.

Type *Non-functional*
The change is related to QoS of the job management service.

Frequency *Often*
The different kind usage of the grid predicts that the adaptation will happen often.

Anticipation *Foreseen*
The Grid is being used by number of users with different requirements therefore the variation in execution time of jobs is the norm.

Mechanism. The mechanism for the self-optimization adaptation is modifying the size of job queues and other relevant properties of the computational site.

Type *Parametric*
The parameters of the CE are changed.

Autonomy *Autonomous*
In this use case there is no outside intervention.

Organization *Centralized*
The adaptation is performed by a single centralized component.

Scope *Local*
In our case the adaptation only involves changes in CE component.

Duration *Short*
The amount of time required to reconfigure the system should be short. Reconfiguration impacts the availability of (some of) the system's services.

Timeliness *Guaranteed*
Torque based CE adapts the queue size immediately, albeit it needs to hold in jobs that we submitted earlier and will now be over the limit.

Triggering *Event-trigger*
The adaptation is triggered when number of jobs in CE queue reaches certain threshold.

Effects. System adjusts the number of jobs that it allows to be scheduled in its queue.

Criticality *Harmless*
The impact upon the system by changing the queue size should be minimal.

Predictability *Deterministic*
There should be guarantees that the system after adaptation will continue provisioning its services.

Overhead *Reasonable*
The torque based CE should have a very little overhear associated with the change of its job queue.

Resilience *Vulnerable*
The system should remain provisioning of its services during the adaptation.

2.4.3 Adaptation

Our proposed solution is to have a control loop for each CE that monitors the number of jobs in the TQ and in the site's batch queue readjusting the queue size when necessary. The initial model (fig. 2.8) should maintain two thresholds: minimum and maximum of the queue size. The maximum should be the maximum of jobs that the site is able to handle before it gets overloaded. As a start, the minimum should be the number of parallel jobs the site can execute or a reasonable multiple. Both values should be subject to adaptation and change as the system evolves. Every batch queue size has a directly proportional tolerance zone associated. When the number of jobs at the site drops below this zone an adaptation might be triggered and the queue size increased. It should be also considered the number of jobs in the TQ is considered; a discrete ratio between the number of jobs to be scheduled and the size of the batch queue. The concrete model is yet to be experienced at short term.

In case of neuGRID, the CE is LCG-CE[12] which is based on torque[13]. Adjusting the queue size in torque has very little impact on the running system, hence we can often modify it. However, there might be different implementations of CE in other gLite deployments, in which queue size change has a more significant impact. In that case a different approach will be developed, for instance by setting an artificial threshold on the queue size and by adjusting the WMS job scheduling as well.

2.4.4 Experimental Setup

For the concrete experimentation, as it was said previously, the problem is that the SALTY test WMS is connected to the CEs that are part of the production environment of neuGRID. Nevertheless, a specific queue will be created on these CEs, in order to minimize the impact of the SALTY experimentations on the production infrastructure. This queue will have a small value for the maximum queue size threshold.

[12] `https://twiki.cern.ch/twiki/bin/view/EGEE/LcgCE`
[13] `http://www.clusterresources.com/products/torque-resource-manager.php`

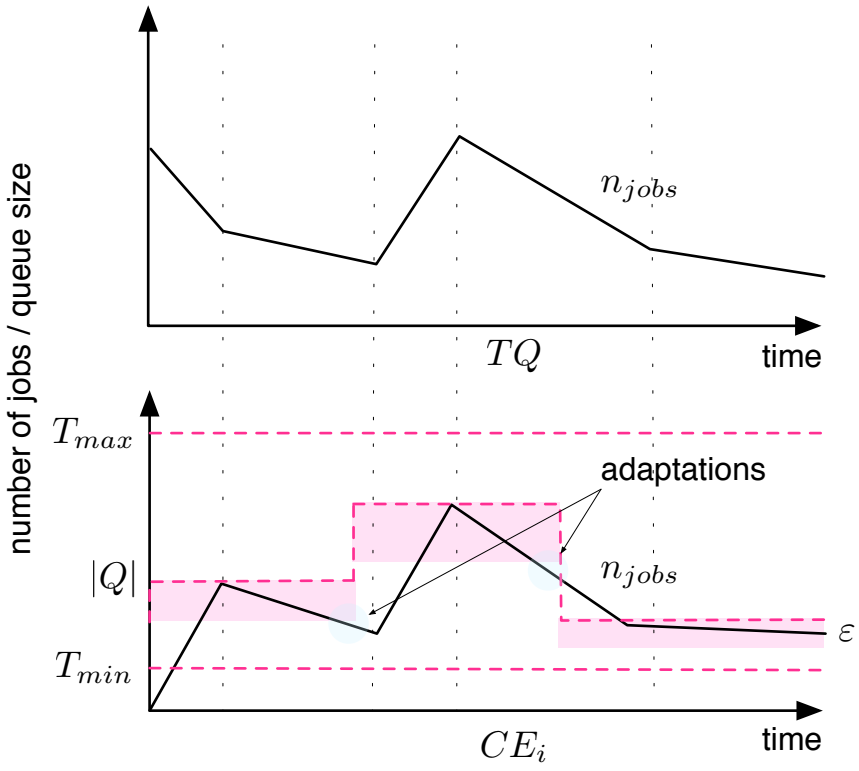


Figure 2.8: CE queue size adaptation model

2.4.5 Remarks

This particular scenario applies mostly in the very specific situation as was the data challenge described in 2.1.3 - in a small environments with lot of sudden work.

2.5 Scenario 1.3 - Job Failures

2.5.1 Objective

Job failures[14] can be divided into two categories: the one where the failure is caused by an application specific problem and the other where it is due to a problem in the Grid middleware. The first category includes invalid job descriptions, application software "bugs" or invalid input data. The cause related to the middleware may be for example some unresolved library dependencies that lead to systematic failures on some jobs. Indeed a job expresses its requirements in a specific JDL file, but there is no fine-grained manner to express precise library dependencies. Therefore a job might be scheduled to run on a WN that does not satisfy the actual job library requirements. The larger the grid considered, the more critical this issue is, as heterogeneity and possible incompatible configurations are more likely in large systems.

Identifying the exact cause of a job failure requires extensive expertise and debugging skills. Furthermore, coordinated investigation over multiple administrative domains is

[14] A failed job in gLite context is considered to any finished job with an exit code other than 0.

often needed in Grids. To address this problem without resorting to a costly human intervention, it is possible to collect statistics to identify recurring source of failures. Although it does not provide insight on the exact reason for the failure, it may be sufficient to avoid situations that are known to fail.

2.5.2 Classification

Below we provide an analysis of the described adaptation in the context of the modeling dimensions from appendix A:

Goals. The self-monitoring subsystem shall increase the number of information that are available in case a job has failed.

Evolution: *Static*
In this scenario only a single goal is considered and it does not change within the lifetime of the system.

Flexibility *Rigid*
The goal is prescriptive.

Duration *Persistent*
The goal is valid throughout the system's lifetime.

Multiplicity *Single goal*
In this scenario we consider a system having only one goal.

Dependency *N/A*
In this scenario we consider a system having only one goal.

Change. The adaptation is triggered when a result of job execution is a failure and when an already failed job is submitted again.

Source *Internal*
The source of job failure is internal to the system[15]. The source of already failed job entering the system is external to the system.

Type *Non-functional*
The change is related to QoS of the job management service.

Frequency *Often*
Job failures do occur on a regular basis as well as there are likely to be already failed jobs retried.

Anticipation *Foreseen*
Although overloads are undesirable, they should be expected hence the overload of WMS (during data challenges in particular) should be considered as foreseen.

Mechanism. When a job has failed during its executing, all possible context information are stored into the knowledge base. When a job that is presented in the knowledge base enters system again its JDL file is extended using some monitoring policies so more information in case of additional failure can be gathered.

[15] it is not easily distinguishable whether the failure has been caused by the job itself or by the middleware so we will consider the source to be the middleware

Type *Parametric*
The parameters of either JDL or knowledge base are changed.

Autonomy *Autonomous*
No outside intervention.

Organization *Centralized*
The adaptation is performed by a single centralized component.

Scope *Local*
Adaptation only involves changes the knowledge base and JDL file

Duration *Short*
The amount of time required for both mechanisms should be rather short.

Timeliness *Guaranteed*
The time need for performing the adaptation both in JDL manipulation and failed job interception can be guaranteed.

Triggering *Event-trigger*
Either by job failure or by already known failed job entering the system again.

Effects. Modified JDL file and or new record in the knowledge base.

Criticality *Mission-critical*
If the change of the JDL file fails it might prevent the job from the actual execution. If failed job does not ended up in the knowledge base the scenario will get compromised.

Predictability *Deterministic*
There should be guarantees that the system after adaptation will continue provisioning its services.

Overhead *Low*
The overhead of modifying the JDL file is rather small as well as should be the getting the notification about job failure and update of the knowledge base.

Resilience *Resilient*
WMS should not be affected during these adaptations.

2.5.3 Adaptation

A first practical approach consists in building a self-monitoring subsystem (cf. Figure 2.9) that gathers information relevant to job failures and indexes them in a knowledge database with their job type (i.e. the full value of the `Executable` directive in the JDL file). It can then be queried to decide some adaptations based on gradual information about failures as well as statistics such as job executable against failure rate.

For example, when a job type that is already in the knowledge base is submitted, the system can adapt the job's JDL according to some specified policies. Technical studies show that this can be achieved by developing a plug-in for the request delivery module in the WMProxy [2]. This allows for experimenting with the job submission and extending available failure information.

Extensions can be addition of standard output, error and core dump files into the job output sandbox that is retrieved by the WMS or executable wrapping with some tracing utility such as `strace`.

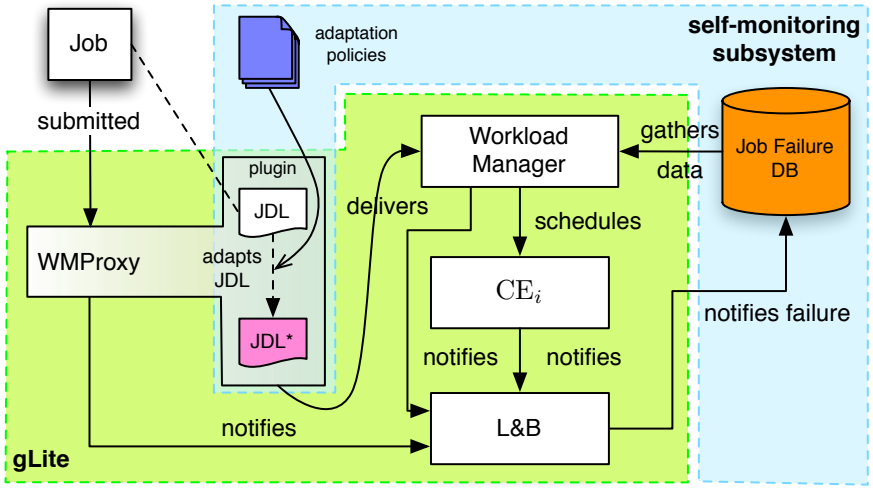


Figure 2.9: Job failure self-monitoring subsystem

Another very useful planned experiment is to verify whether the particular job type fails on all CEs or only on a certain subset. To implement this, an adaptation could modify the JDL to blacklist one or more CEs.

To populate the database, the self-monitoring system interacts with the L&B service to get all reported[16] failures. Unless explicitly stated in the JDL, there is very little information available upon job failure when the cause is not properly identified by the middleware, *i.e.* usually only the exit code, and that makes root cause analysis very difficult. Therefore, the other part of the self-monitoring subsystem gathers more relevant information by interacting with the WMProxy and by monitoring all new incoming jobs.

Further we can envision extending the monitoring subsystem with the ability to cluster failures by input data.

2.5.4 Experimental Setup

The experiments for this scenario will be done in the MAAT-G testing environment. We will execute a series of jobs with verified behavior: either successful execution of failed one and experiment with different adaptation polices and JDL modifications. This should not put too much stress to the underlying environment.

2.6 Scenario 1.4 - CE Black Hole

2.6.1 Objective

Under certain circumstances a CE might malfunction and start to fail all scheduled jobs for some unknown reason. Since it fails all jobs immediately, it will process its queues very quickly hence becoming a *black hole* in the Grid as it will eat all newly incoming jobs that are matched to its configuration. This scenario is not directly linked to failures

[16] There might be jobs that disappeared because for some reason the middleware lost track of them. However, these jobs will not be detected as failed.

observed during the data challenge, but it is a well-known issue in the gLite middleware [4].

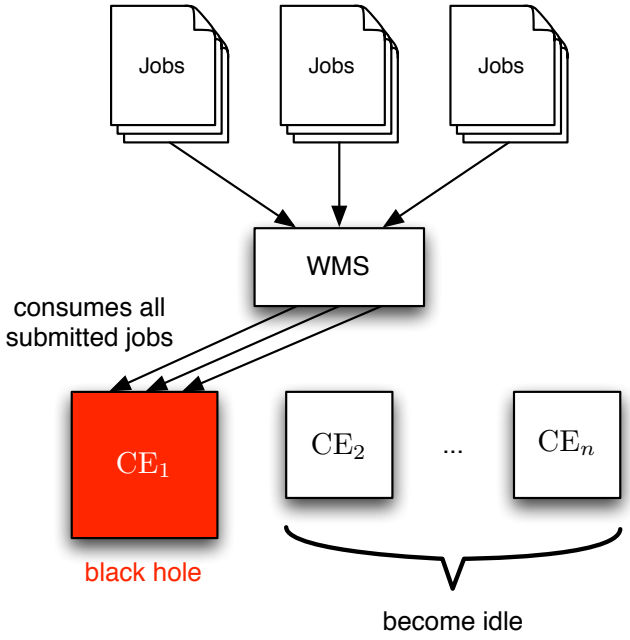


Figure 2.10: CE black hole

2.6.2 Classification

Below we provide an analysis of the described adaptation in the context of the modeling dimensions from appendix A:

Goals. The self-healing subsystem shall put out of operation any computational site that has been detected as black holes.

Evolution: *Static*
In this scenario only a single goal is considered and it does not change within the lifetime of the system.

Flexibility *Rigid*
The goal is prescriptive.

Duration *Persistent*
The goal is valid throughout the system's lifetime.

Multiplicity *Single goal*
In this scenario we consider a system having only one goal.

Dependency *N/A*
In this scenario we consider a system having only one goal.

Change. The adaptation is triggered when a black hole pattern is detected.

Source *Internal*
The change is internal as it occurs in a CE which is part of the middleware.

Type *Non-functional*
The change is related to QoS of the job management service.

Frequency *Rare*
During the normal grid operation it does not happen very often that a CE will dysfunction and become a black hole.

Anticipation *Foreseen*
Although black holes are undesirable, they should be expected.

Mechanism. The mechanism of the adaptation is to eliminate the malfunctioned CE as soon as possible by putting it to a drain mode.

Type *Combinational*
On the one hand, the parameters of the CE service are changed. Putting CE to a drain mode is done in CE it self by changing its configuration. On the other hand, it is also a structural change to the rest of the system, as the CE will be finally removed from the available resources at the WMS level (see adaptation part).

Autonomy *Assisted*
A system administrator is eventually required to take an action.

Organization *Decentralized*
It depends at the final implementation, but in general there is no correlation between CE and therefore it is decentralized.

Scope *Global*
The adaptation involves putting CE into a drain mode which in consequence will affect the WMS as well.

Duration *Short*
The amount of time required to reconfigure the system should be short.

Timeliness *Best-Effort*
Once a CE is put into a drain mode, it will stay there until a system administrator does not change it.

Triggering *Event-trigger*
The adaptation is triggered when a black hole pattern is detected.

Effects. System disables the malfunctioned computational site.

Criticality *Mission-critical*
If the adaptation fails to disable the malfunctioned CE, it will continue to fail all the jobs submitted to the infrastructure.

Predictability *Deterministic*
There should be guarantees that the system after adaptation will stop advertising itself to the ISM.

Overhead *Small*
Putting CE into drain mode is a quick action that does not require almost any resources.

Resilience *Vulnerable*
The CE itself will stop working, but this is actually expected. On the other hand, the job submission mechanism should continue to work.

2.6.3 Adaptation

The self-healing adaptation in this case involves a control loop that monitors execution time, IO activity using low level operating system probes and results of job execution using the L&B service or CE's log. When it observes the black hole pattern – a series of jobs with very short execution time and low disk activity – it will put the CE into a drain mode. Drain mode will be reported back to the ISM (Information Supermarket) and after several minutes, the WMS will no longer submit jobs to it. This will also be propagated to system administrators who should take a closer look at the problem and are responsible for bringing the site back up and running.

In this scenario there are multiple options on the concrete loop deployment. For example there may be one control loop per CE or one *master* control loop that manages all CEs in the infrastructure. In the former option another loop will be required to manage loops together with CE life-cycles, so when a new CE joins a new properly configured loop is deployed into the system and vice-versa. The different pros and cons of these approaches are to be further experimented and one of the aims of the SALTY framework is to facilitate and capitalize such experimentations.

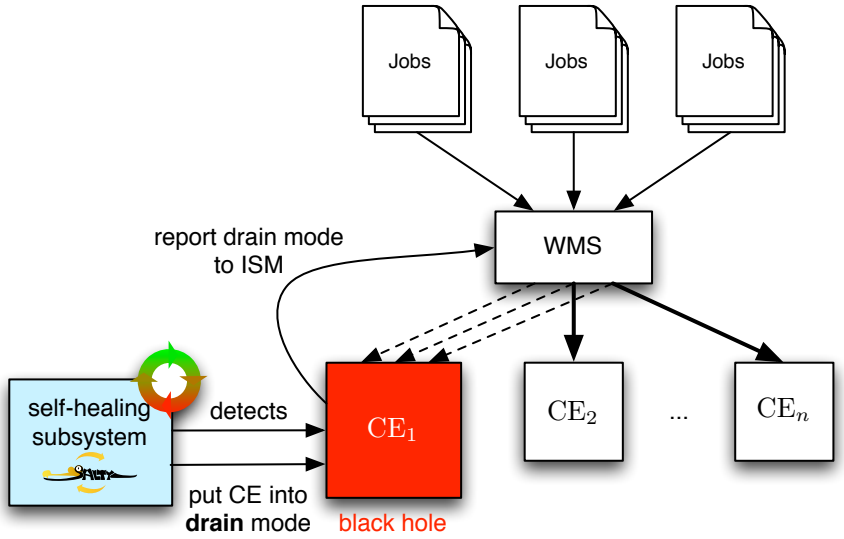


Figure 2.11: Self-healing adaptation

2.6.4 Experimental Setup

For this specific case, having a specific queue on the production neuGRID CEs will not be sufficient. In this particular case, a specific CE will have to be installed and configured to be a black hole.

2.6.5 Remarks

An extension should be that there are more then one mechanism to disable the CE. Currently the scenario only includes the one that sets CE into drain mode. This is however an indirect action as the CE itself needs to propagate it to the ISM. If from some reason

the CE is not able to do that, some *hard* mechanism should be available to guarantee the effect, even for a price of a system shutdown or a similar action.

CHAPTER 3

Scenario 2 - Desktop Fusion Self-Configuration

3.1 Context

3.1.1 Introduction

Scientists' applications need more and more storage and computing resources. Grid Infrastructures can provide such facilities. However, installing and configuring all applications on machines may be seen as a burden, especially for Grid middleware. Providing easy-to-use applications, with no installation and configuration steps is clearly an advantage, and it's the goal of *Desktop Fusion*.

3.1.2 Desktop Fusion

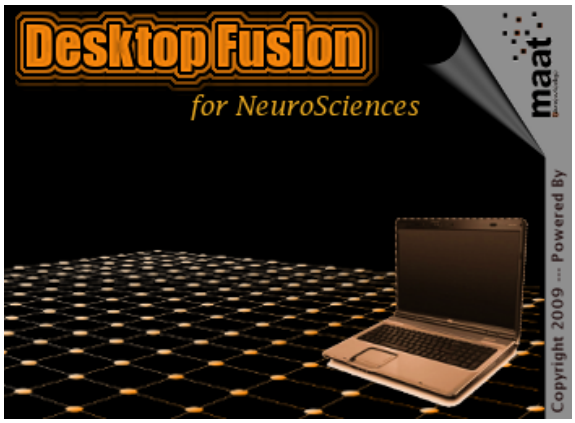


Figure 3.1: Desktop Fusion splash screen

.

Desktop Fusion is an integrated new technology which allows for remote execution of applications. The technology chosen to achieve this is the Open Source version of NX so call FreeNX. The latter provides encrypted and optimized access to remote applications thus allowing researchers to run specialized viewers and to interact with the Grid directly from their desktop (fig. 3.2).

3.2 Scenario 2.1 - Dynamic Load Balancing

3.2.1 Objective

Desktop Fusion provide users with remote access to applications which may be more of less compute-intensive. Therefore, several users using *Desktop Fusion* at the same time

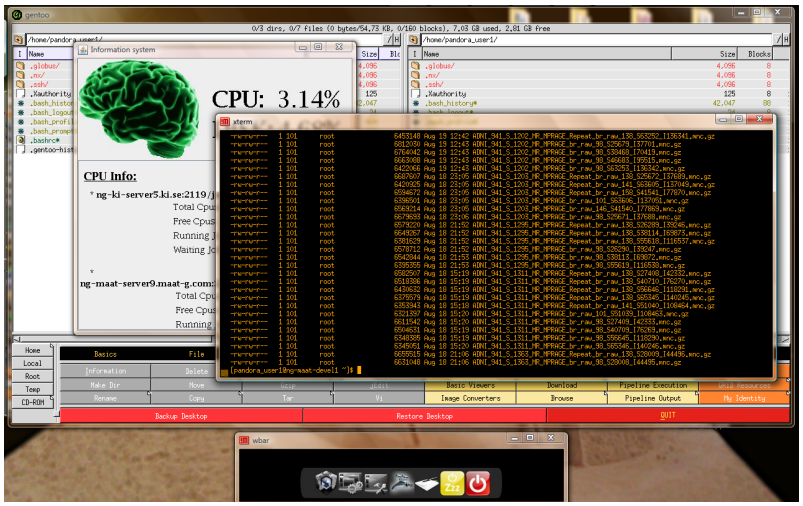


Figure 3.2: Desktop Fusion screenshot

.

could rapidly result in a poor Quality of Service (QoS).

The infrastructure needs to be adapted according to the number of connected users and applications they are using. Taking into account low level Operating System information such as memory and CPU usage is necessary as well in order to have a better idea of current load of the system. Thus, when a situation of overload will likely happen, a new *Desktop Fusion* server will be deployed. It will be configured as part as the load-balanced alias, so that it will be completely transparent for users. This could handle the case where more and more users are connecting to Desktop Fusion, or if a Desktop Fusion server goes down for some reasons.

3.2.2 Classification

Below we provide an analysis of the described adaptation in the context of the modeling dimensions from appendix A:

Goals. The system shall allow for automatic maintenance of some minimal Quality of Service for the users.

Evolution: *Static*
In this scenario only a single goal is considered and it does not changes within the lifetime of the system.

Flexibility *Rigid*
The goal is prescriptive.

Duration *Persistent*
The goal is valid throughout the system's lifetime.

Multiplicity *Single goal*
In this scenario we consider a system having only one goal.

Dependency *N/A*
In this scenario we consider a system having only one goal.

Change. The adaptation is triggered when system resource usage or number of connecting users reach certain thresholds.

Source *External*
The change will be the deployment of a *Desktop Fusion* server on a new machine.

Type *Non-functional*
The change is related to QoS of the Desktop Fusion service.

Frequency *Occasionally*
Although server crashes should happen rarely, the number of connected users varies in the time, so does the number of required Desktop Fusion servers.

Anticipation *Foreseen*
Although server crashes are not predictable, the number of connected users can be easily monitored, and the change can be applied while QoS is still acceptable.

Mechanism. The mechanism for the self-healing adaptation is to deploy and configure new *Desktop Fusion* servers.

Type *Deployment*
New *Desktop Fusion* servers need to be deployed.

Autonomy *Autonomous*
In this use case there is no outside intervention.

Organization *Centralized*
The adaptation is performed by a single centralized component.

Scope *Global*
In our case the adaptation involves deployment of a new machine which will be integrated in the system.

Duration *Average*
The amount of time required to deploy a new machine in the system has to be measured, but it should not be that long.

Timeliness *Best-Effort*
Although the time needed to deploy a new machine is quite constant, some differences could appear, depending on the network usage during the adaptation.

Triggering *Event-trigger*
The adaptation is triggered when resource usage or number of connected users reach certain value.

Effects. Deployment a new machine doesn't impact the system availability.

Criticality *Mission-critical*
It is mission-critical as we cannot guarantee the Quality of service if the deployment of the new machine fails.

Predictability *Deterministic*
There should be guarantees that the system after adaptation will continue provisioning its services.

Overhead *Reasonable*
There should be a mechanism to ensure the system does not constantly reconfigure itself. This is realized by ensuring that the adaptation is triggered only if there are significant changes in the monitored data over a pre-specified period of time.

Resilience *Resilient*
Service availability is not impacted while being reconfigured.

3.2.3 Adaptation

The need of an extra Desktop Fusion server is detected when the number of connected users, CPU usage (per server) exceed certain threshold values.

When the loop detects that a threshold is exceeded, the adaptation mechanism will deploy a new machine, with Desktop Fusion, and will integrate it in the load-balanced alias, so that users can access the new machine in a completely transparent way.

3.2.4 Experimental Setup

The Desktop Fusion system is based on the NX technology [1] and more particularly on the FreeNX implementation [2]. This later contains already a load-balancing functionality: basically, a list of nodes is maintained on the master FreeNX server. This list will be modified by the SALTY system.

Moreover, new Desktop Fusion servers will have to be concretely deployed. All the neuGRID infrastructure will be migrated from Xen to XenServer soon and proper Desktop Fusion virtual machines will be created. This will allow the Salty system to deploy these later contacting the XenServer system.

3.2.5 Open Questions

It is to be investigated whether the list of FreeNX nodes can be modified on the fly without having to restart the master node. Moreover, in this simple version of the scenario, nothing is specified in the case the number of users decrease and uninstallation of servers should be expected. As the project progresses, we envisage to port and adapt the similar solutions that are going to be developed for the WMS overload scenario on the gLite middleware.

[1] See `http://www.nomachine.com/`.
[2] See `http://freenx.berlios.de/`.

CHAPTER 4

Scenario 3 - Self-Adaptive Enterprise Service Bus

This scenario builds on the Petals *Enterprise Service Bus* (ESB). Petals is a distributed platform in which *services* are deployed and invokes each others in the context of integration processes. Since this ESB is based on a *Service-Oriented Architectures* (SOA), the controlled elements of Petals that are going to be involved in the SALTY adaptation processes are services. These services are involved at different levels: *i)* for optimizing the Petals platform and *ii)* for preserving from inconsistencies in service workflows deployed on Petals.

4.1 Context

In this section, we introduce the motivation of the scenarios dealing with Petals ESB and the issues coming from its distributed nature. Nowadays, the amount of Web services available on the Internet is growing continuously and the next challenges consist in combining them in order to build complex services, while facing services discovery and matching issues. In a SOA ESB context, a common issue identified relates to *service composition* and *orchestration*. In particular, a key challenge consists in invoking services within a dynamic context, that requires adaptations at runtime.

We consider for the time being a number of services within a distributed bus, indexed in a distributed registry. Two scenarios have been developed: the first one deals with adaptability of the distributed registry, while the second one addresses a concrete use case that illustrates workflow adaptation needs in an ESB.

These use cases aim at proving that distribution and adaptability challenges identified in a ESB can be solved thanks to the SALTY framework.

4.1.1 Introduction

Petals ESB is an Open Source (LGPL License) ESB hosted by the OW2 middleware consortium[1]. It is built on top of agile technologies, such as:

- The *Java Business Integration* (JBI) v1.0 specification, which is the Java standard for enterprise application integration. The Petals ESB has been certified by SUN Microsystems as a valid JBI implementation.
- The FRACTAL software component framework provided by the OW2 consortium[2]. Fractal is a modular and extensible component model that can be used with various programming languages to design, implement, deploy, and reconfigure various systems and applications, from operating systems to middleware platforms and to graphical user interfaces. From the Petals ESB point of view, all the container services (service registry, message router, message transporter, discovery, etc.) are

[1] See `http://petals.ow2.org`.
[2] See `http://fractal.ow2.org`.

therefore implemented as FRACTAL components. This is a major feature, which allows core developers to tune a Petals ESB distribution by choosing the software components to be used for specific needs.

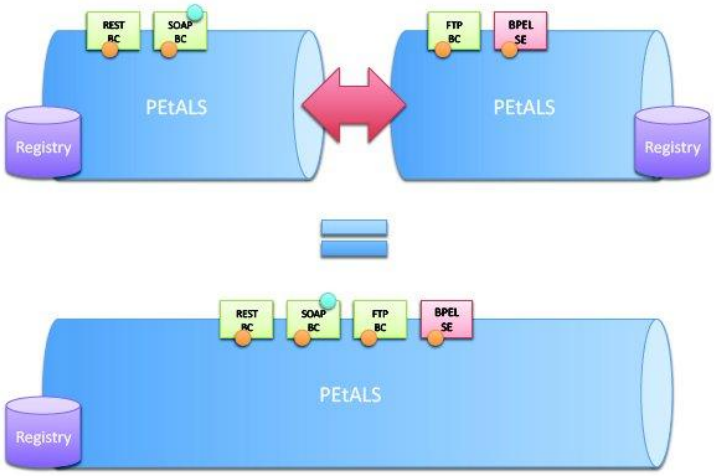


Figure 4.1: The distributed Petals ESB.

The main Petals ESB feature is the extension of the JBI specification to provide a distributed support for JBI platforms (cf. Figure 4.1). Several Petals containers deployed on several nodes are equivalent to a single unified Petals ESB container. This transparent distribution approach ensures that all the services remain accessible just as in a typical standalone JBI environment. While other JBI implementations provide a distributed approach by connecting their JBI containers with the use of JBI Binding Components and complex configurations, Petals ESB natively supports this feature without imposing any additional configuration. This distributed behavior is enabled by the following software components:

- The *technical registry*. The Petals JBI services, endpoints, interfaces, WSDL descriptions, and container location (physical network address) are stored in the technical registry. This registry is used by the Petals ESB container to register services and to route the JBI messages to the right endpoint. The registry entries are replicated among all the Petals ESB nodes using a *Distributed Hash Table* (DHT) over a multicast channel. This is equivalent to data-flooding between registries—*i.e.*, when an entry is added to the registry, the data is forwarded to all the network registries. In this way, all the registries have a complete view of the services hosted by all the containers.

- The *message transporter*. This layer is not defined in the JBI specification. Its role is to exchange JBI messages between containers. In a standard JBI implementation, the *Normalized Message Router* (NMR) gets the local endpoint reference from the local registry and sends the message to local JBI endpoint. In the Petals ESB approach, once the endpoint is retrieved from the local registry, the message and the endpoint reference are sent to the transport layer, which is in charge of delivering the message to the JBI endpoint independently of the container location (local or remote).

Petals ESB is not only a standard JBI container, but in addition provides various frameworks, components and tools for extension, service integration, management and monitoring purposes:

1. The *Component Development Kit* (also named CDK) is a software framework which abstracts all the JBI related API and provides an easy way to develop high performance JBI components (*Service Engines* [SE] and *Binding Components* [BC]) with a small set of Java classes.
2. All JBI components are based on the previously cited CDK framework. The actual collection of Petals JBI components is composed of binding components (provide connectivity to/from external services), such as SOAP, FTP, JDBC, XMPP and service engines (provide internal technical services) like BPEL, BPMN, XSLT, EIP, or SCA.
3. The WebConsole is a Web GUI used to monitor and manage the Petals ESB containers. This tool provides a single access point to monitor and manage all the distributed containers.

Finally, we propose an additional tool for service monitoring that must be enhanced according to SALTY works. This monitoring bus, based on the WSDM specification[3], should be able to act as a sensor for technical and functional information retrieval, and as an actuator to reconfigure the deployed services. Therefore, we plan to integrate in the MAPE loop as dedicated touch-points.

As a summary, the Petals ESB consists in a set of containers (or nodes) within a topology that are deployed on the Internet. Each node contains a set of services and hosts its own registry. The whole architecture, composed of several nodes, provides a distributed architecture for complex services integration possibly implying their orchestration.

4.1.2 Petals ESB Registry

The actual implementation of the Petals ESB registry consists in a *Master/Slaves* solution. One of the nodes contains the unique master registry and other nodes contain slave ones, that have to request the master registry in order to update their index.

However, this solution can raise some consistencies when the node containing the master registry falls down. No update can be made anymore and registries become desynchronized.

4.2 Scenario 3.1: Self-Organization of the ESB Distributed Registry

4.2.1 Scenario Description

In this scenario, we consider a topology of some nodes composing the Petals ESB middleware. We define a master registry and deploy several services on each node. Based on this first architecture, we add a conceptual layer corresponding to the SALTY framework, implemented thanks to SCA *Service Engine* (SE), one sensor able to detect the master node failure, and one effector able to update the role of registries (from slave to master).

4.2.2 Objective

This scenario (cf. Figure 4.2) aims at providing a solution of adaptability for the Petals ESB distributed services registry.

[3]Web Services Distributed Management: `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm`

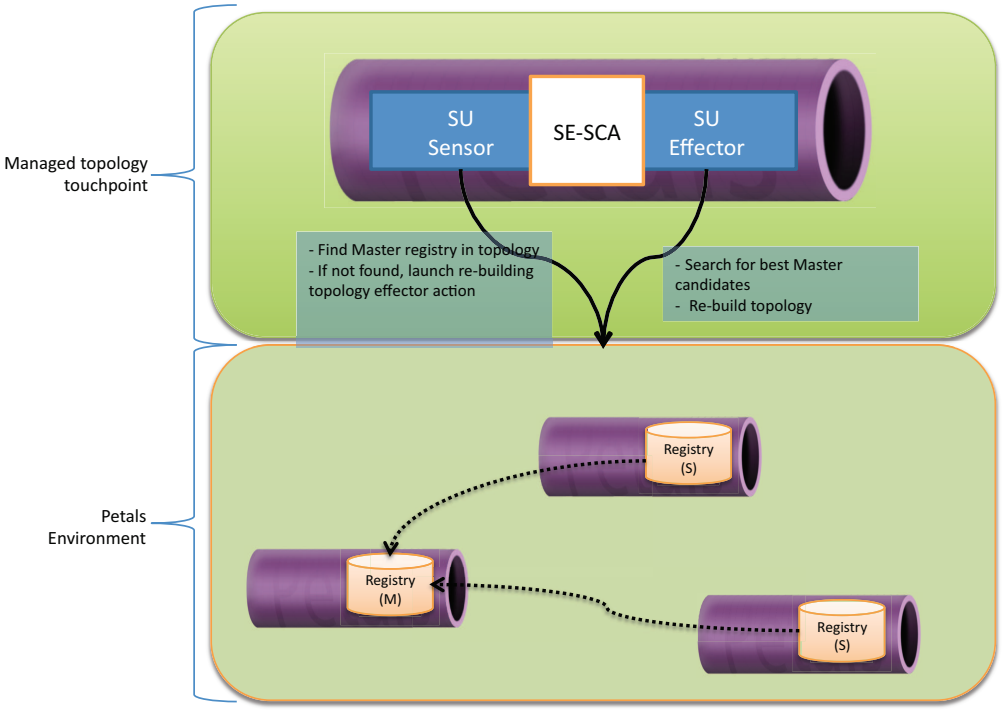


Figure 4.2: Registry consistency scenario.

The service registry indexes all available service endpoints and is dynamically updated according to services (un-)installations occurring in the ESB.

One of the Petals ESB specificities is its distributed architecture, composed of nodes disseminated over a large network, configured according to a given topology. Then, the registry mechanism consists in a master/slaves architecture pattern: one node contains the master registry and the other ones contains slave registries. The master registry receives queries from slave ones for updates and thus maintains the current state of the ESB services.

This mechanism can be prone to instability when nodes fail and, in particular, if it occurs to the node containing the master registry. The SALTY framework can address this weakness by adding an upper monitoring and control layer, based on *Service Component Architecture* (SCA) sensors/effectors, allowing to know the master registry is available in the current topology of nodes and if not, launching a recovery process to rebuild a consistent distributed registry. Some ways can be envisaged in order to perform this task:

- rebuilding the whole topology of registries and switching a slave registry to a master one,
- selecting a predefined registry to become the new master one.

This adaptation task can be engaged if an ESB node crashes. Then, it is no longer in the running state—*i.e.*, all the components and services that were deployed and run in this ESB node are also no longer in the running state. The whole software system running over the Petals ESB can be affected; part or all the services and functionalities provided to the end-user/customer can be disrupted.

4.2.3 Classification

In this first scenario the adaptive task corresponds to a self-healing subsystem which purpose is to have always a master node with an up-to-date registry.

Goals. The goal is to preserve the services index consistency in cases when Master registry crashed.

Evolution: *Static*
The goal consists in keeping the topology consistent and it will not change.

Flexibility *Rigid*
The goal must be reached. There is no uncertainty associated with this goal.

Duration *Persistent*
The topology consistency is mandatory throughout the ESB platform's lifetime.

Multiplicity *Single goal*

Dependency *N/A*

Change. The adaptation is triggered when it is detected that no master registry is available in the topology.

Source *Internal-middleware*
Since one existing node and the topology is updated, changes occur in the infrastructure.

Type *Technological*
The topology of the middleware reflecting the actual physical nodes is changed.

Frequency *Rare*
Nodes are quite stable and system failures come generally from hardware.

Anticipation *Foreseen*
Although it does not happen very often that a master node would fail, it can happen and therefore it is anticipated.

Mechanism. The mechanism consists in reconfiguring the ESB by dynamically changing the role of a registry and re-synchronizing service indexes.

Type *Structural*
The mechanism consists in changing the structure of the system.

Autonomy *Autonomous*
No outside intervention is required during the adaptation task mechanism.

Organization *Centralized*
We consider a single sensor for the overall architecture. A distributed solution should be found for a better solution, by managing topology probing thanks to sensors present on each node of the overall topology.

Scope *Global*
The whole topology is supposed to change.

Duration *Short*
Some minutes can be expected to update topology.

Timeliness *Best-Effort*
The time to perform this task is depending on the time to find a valid candidate node or to restart a node.

Triggering *Event-trigger*
The task occurs when some exchanges in the ESB are not resolved anymore from some service endpoints. However, we could envisage a time-triggered solution as a separate process implementing a failure detector and probing for nodes at a given frequency.

Effects. The topology of the bus changes.

Criticality *Mission-critical*
There is no guarantee to find an available candidate node. In this case, the system will be left without synchronized registry and therefore non-functioning.

Predictability *Deterministic*
In case of success, nominal activity is expected. In case of failure, the system can perform exchanges routing for the remaining available services.

Overhead *Reasonable*
The reconfiguration action should not alter the system.

Resilience *Vulnerable*
If some registry updates are done during the reconfiguration, these updates will not be taken into account. However, some solutions should be found in order to face this issue: either thanks to transactional processes but does not seem to be suitable for very large distributed systems, either thanks to some neighborhood probing and step by step synchronization.

4.2.4 Adaptation

According to systems self-management definitions, this adaptation task is related to Self-Healing [8]. A change is detected thanks to original topology description. If this change corresponds to master registry disappearing, the effector launches an adaptation process in order to find an available registry, to change its role and to probe effective endpoints available on the Petals architecture. Four types of self-repair actions are therefore needed: monitoring actions, analyzing actions, planning actions, and execution actions.

Monitoring actions The targeted system needs to be monitored in order to collect information that will be analyzed to detect when the ESB node crashes. This monitoring can be done, by default, directly by the end-users (they will ask for support if they can no longer access to their services). This is, of course, insufficient and even inefficient. Here, an automatic monitoring system is required. This monitoring system must collect monitoring data related to the ESB node itself (including the NMR), to the JBI components, and to the *Service Unit* (SU) and *Service Assembly* SA. As a consequence, the Petals ESB must be equipped in order to provide the needed monitoring probes (*i.e.*, the needed sensors).

These probes consist in a SALTY SCA framework and can be executed inside or outside of the ESB. They could also be implemented in a centralized way (rather outside

the bus) or a distributed way (rather inside the bus). We could argue that probes executed outside the ESB would be smarter in order to report crashes without crashing themselves. However, this implies to envisage meta-probes implementation in order to verify first-layer probes activity, and so on. Then, we envisage embedded SCA probes consisting in sensors deployed on each node of a Petals ESB topology, implementing at least a "*Heartbeat*" pattern. This pattern implies a time-triggered adaptation. For more reactivity, we could imagine an event-triggered one by implementing loss of messages management, implying synchronization issues that can come from master node crash.

Note also that two types of monitoring can be done, either a live one or a *post mortem* one. A live monitoring is done when the targeted system is running (when the end-users use it). We clearly talk here of real-time system. On the other hand, a *post mortem* monitoring system can provide off-line monitoring data. A simple *post mortem* example is a monitoring system based on logs: the monitoring system processes the logs during the night and produces monitoring data in the morning. It is obvious that introducing a self-repair mechanism based on a live monitoring is much more interesting for the targeted system than introducing the same mechanism over a *post mortem* monitoring.

Analyzing actions Analyzing actions can be trivial or much more complex. Their complexity depends on the available monitoring information collected from the probes. Indeed, if all the controlled elements have their own "*Heartbeat*" probe then the analyzing actions only need to check if the controlled elements are (still) alive. This *Heartbeat* mechanism should consist in a event-based notification sent to the control system, in an active way, or a ping-like operation exposed on each controlled system in order for the control system to check their availability, in a passive way from the controlled system point of view.

On the other hand, if only indirect monitoring information is available (*e.g.*, the average response time during the last day, the number of requests received without any information on the responses sent) then the analyzing actions will be much more complicated even impossible.

Planning actions In case of an ESB node crashes, the planning process must request the following execution actions:

1. the cleaning of the computer in which the crashed ESB node was running (checking that the Java Virtual Machine really stopped, removing the Petalslock file, etc.)
2. the installation and deployment of the corresponding ESB node components (*i.e.*, the NMR, the registry, etc.),
3. the installation and deployment of the corresponding JBI components (*i.e.*, the BC and SE),
4. and the installation and deployment of the corresponding SU and SA.

Note that the planning process produces an ordered list of actions. Also note that if a "*check ESB node current state/status*" mechanism exists, then the planning process must also request it once the all the ESB elements are installed and deployed.

Execution actions Handling an ESB node crash implies to own effectors at four different levels:

- Computer level: to clean it, to install an ESB node in it, and to launch/start/activate/instantiate an ESB node,
- ESB node level: to clean its content,
- BC/SE level: to clean their content (*i.e.*, to reset them), and to install/activate JBI components,
- SU/SA level: to clean their content (*i.e.*, to reset them), and to install/activate SU and SA.

Note that effectors at computer level should also be able to (statically or dynamically) update an ESB node, to shutdown/deactivate a running ESB node, and to uninstall an ESB node. Similarly, effectors at BC and SE components level should also be able to (statically or dynamically) update, to stop/shutdown/deactivate, and to uninstall JBI components; while effectors at SU and SA levels should also be able to (statically or dynamically) update, to stop/shutdown/deactivate, and to uninstall both SU and SA.

4.2.5 Experimental Setup

The experimental setup will be initially done on dedicated testing servers at EBM Websourcing. It will consist in using a set of virtual servers in which Petals nodes will be installed.

JBI components will be used upon these Petals nodes, to simulate a realistic middleware environment:

- SOAP component: able to provide external Web services to the bus,
- SCA component: able to provide execution environment for SCA services. In particular, 2 kinds of artifacts will be developed at least for this component (a sensor and an effector).

4.3 Scenario 3.2: Self-Adaptation on a Crisis Management Workflow

4.3.1 Use Case Description

In this use-case, we are interested in workflows, which are performed within an ESB and can be reconfigured at runtime. In such ESBs, workflow implementation typically consists in an orchestration of services and a well known standard such orchestrations is WS-BPEL[4]. It consists in defining activities providing *control structure*, *invocation* and *receive mechanisms*, *correlation* and *compensation features*, able to build an execution graph or workflow. This workflow supports several services for working together. They are called *partners* and are defined thanks to their WSDL interfaces.

In our context, an adaptation consists in updating an orchestration by changing parts of its execution graph or its partners. Indeed, we believe that reflective mechanisms can help in facing new orchestration issues coming from the complexity increase of business processes, becoming longer in time thanks to asynchronous mechanisms and being involved in more and more dynamic contexts. We build on a legacy use-case in order to illustrate the key issues of workflow adaptations. It consists in a crisis management for

[4]WS-BPEL: `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`

local authorities, which have to plan for chemical, biological, radiological, and nuclear accident (so called CBRN crisis). This issue, identified in the former ANR SEMEUSE project had some results about dynamic workflows implementing thanks to late binding and semantic match-making extended activities[5]. In this project, we therefore progress on this scenario by addressing the reconfiguration of the workflow itself at runtime, based on non-functional and functional retrieved information.

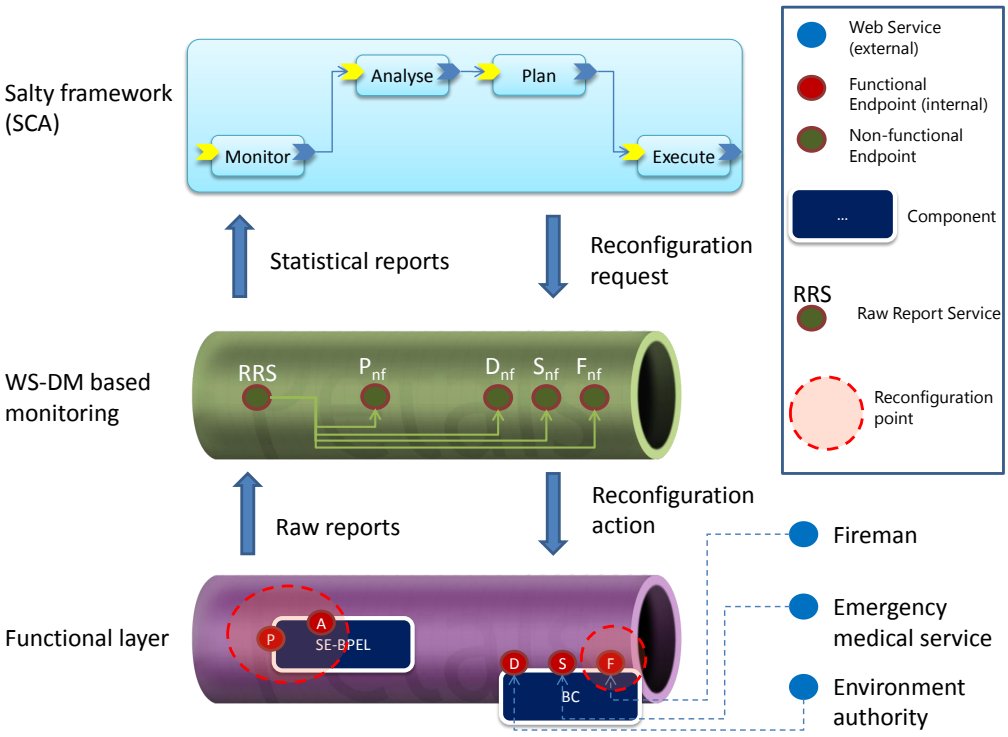


Figure 4.3: Adaptive workflow use case architecture.

The functional architecture depicted in Figure 4.3 describes the involved elements:

- a SOA ESB containing functional services and in particular the orchestration one,
- a monitoring bus based on WSDM, the standard for services management that acts as sensors and effectors,
- and an implementation of the MAPE feedback control loop based on SCA.

Within the context of a SOA ESB, every partner is represented as a service. Thus, according to the use case, the ESB contains the following functional services: Fireman "F", Emergency medical service "S" and Environment authority "D". It contains also the orchestration service called "P" and an administration service called "A". Each service deployment is intercepted within the bus in order to generate a corresponding non-functional service within the monitoring bus. Then, exchanges occurring at runtime in the Petals ESB are intercepted and sent to the corresponding non functional services *via* a "*Raw Report Service*" (RRS). This RRS acts as a broker for reports sent by interceptors. These reports contain exchange information: *latency*, *percentage of successful exchanges* and the *name of the service associated with these informations*. Then, statistical measurements are

[5]ANR SEMEUSE Project: `http://www.semeuse.org`

made all along the lifetime of the system into each non-functional service. We use WS-Notification[6] mechanisms that allow non-functional services to send computed information as notifications. The monitoring part of the MAPE feedback control loop subscribes to these notifications in order to receive these reports.

The MAPE feedback control loop consists in a SCA implementation of reconfiguration policies. Each task of the MAPE feedback control loop corresponds to an aggregation of information retrieved from the services. Chosen reconfiguration is finally executed on Petals ESB thanks to actuators driven by these policies. They are implemented as BPEL processes, which are able to reconfigure the functional BPEL process of the crisis and/or to administrate partners.

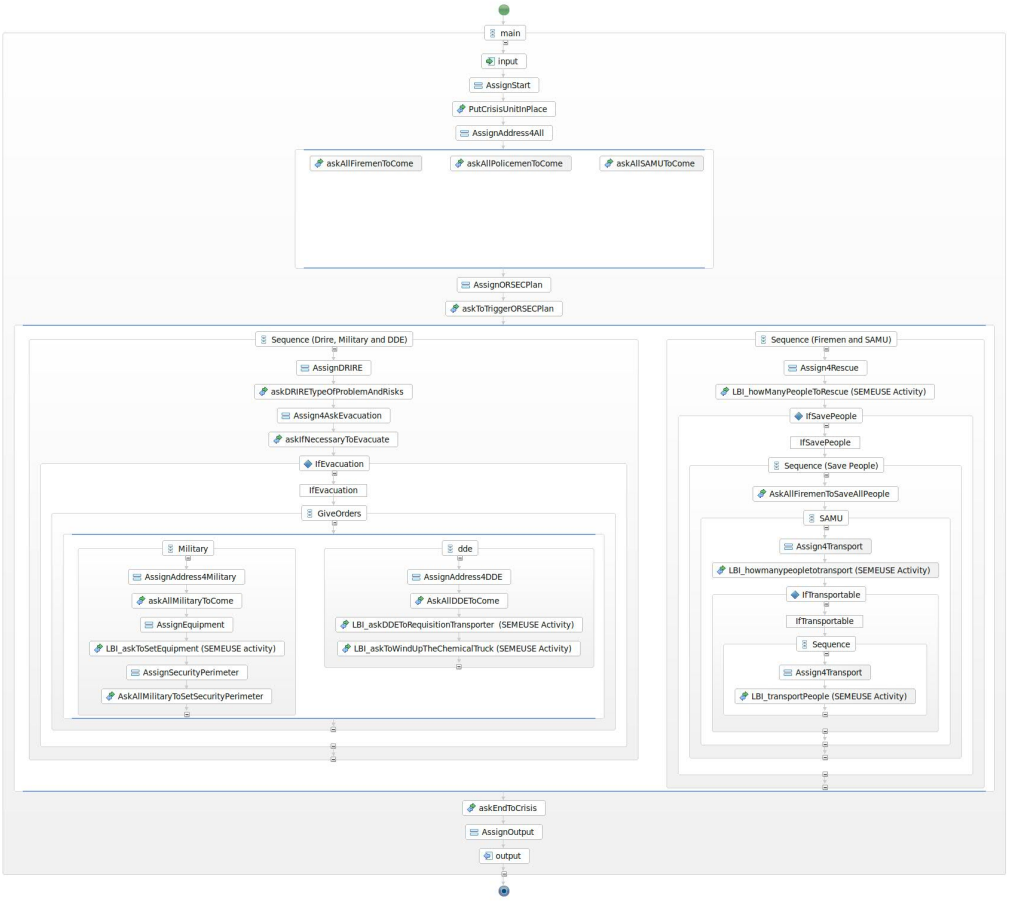


Figure 4.4: BPEL process of NRBC use-case.

In the case of our concrete crisis management example, the workflow to adapt and deployed into the bus is implemented by a BPEL process (cf. Figure 4.4). Partners services are Firemen, Emergency medical service, and Environment authority. Non-functional services deployed on the WSDM bus are able to catch technical information such as their *latency* and *load*. However, we are not only interested in retrieving technical information. Indeed WSDM specification also proposes to catch business information and this feature could be thought in our use-case. In particular, information, such as *firemen heath rate* or *temperature*, *environment authority crisis state report*, for instance can be reported in order to be involved in the adaptation process.

[6]WS-Notification: `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn`

Two adaptations are then possible. The first one consists in taking into account the dynamic aspect of a crisis, faced by the dynamic features of the workflow. WSDM can report changes during a crisis like an explosion for instance (reported by the Environment authority partner). At MAPE level, workflow reconfiguration can be planned in order to add a branch of execution to manage this explosion. This adaptation is located at ”P” service.

A second adaptation takes into account firemen functional information (*e.g.*, temperature, heart rate) and corresponds to services invocation reconfiguration. This peculiar information allow the feedback control loop to detect that firemen are not available anymore and must be evacuated. The MAPE feedback control loop can trigger a reconfiguration process in which firemen partners involved into the workflow are changed: an evacuation business operation can be invoked and/or administration on Firemen services can be used to stop some of them and to start another ones. This adaptation is located at ”F” service.

4.3.2 Objective

This use-case deals with *self-configuring* and *self-protecting* adaptation type. Main aims are to be able to adapt a workflow of services according to a dynamic context. This adaption allows us to define a reactive workflow, remaining operational when changes occur.

We want to leverage from SALTY framework in order to validate technical and business WSDM based services monitoring within the context of an ESB.

4.3.3 Classification

Goals. The goal of this task is to adapt a business workflow according to new needs occurring at runtime.

Evolution: *Static*
The goal remains the same until the crisis is ended, that is to say, the end of the workflow.

Flexibility *Rigid*
The goal must be reached. There is no uncertainty associated with this goal.

Duration *Temporary*
The system continues after the end of the crisis, waiting for a new one.

Multiplicity *Multiple goals*
Two goals are defined: (i) adapting the workflow according to the crisis evolution and (ii) replacing deficient partners by operational ones.

Dependency *Independent*
It can be considered that workflow is going to be updated on different parts according to the goal. However it can be risky to perform both reconfigurations at the same time.

Change. The adaptation is triggered when the Environment authority detects a change in the crisis or when some partners are detected as not operational anymore.

Source *Internal*
Changes occur within the workflow and already defined and known partners.

Type *Functional*
Changes occur on functional aspects of the system (adding a branch to the workflow, replacing a partner).

Frequency *Rare*
We are in a long term workflow context where updates can be considered as rare.

Anticipation *Foreseeable*
This adaptation task must be planned by authorities.

Mechanism. The mechanism consists in using reconfiguration workflows able to update functional one and in using administration features provided by the Petals ESB and the BPEL engine.

Type *Parametric*
Changing partners or updating the workflow can be performed thanks to configurations.

Autonomy *Autonomous*
No outside intervention is needed during the adaptation task mechanism.

Organization *Decentralized*
The WSDM monitoring server is a bus able to be deployed in a distributed manner. Moreover, several components are involved in the adaptation process (from sensors in the WSDM bus to actuators performed on the Petals ESB).

Scope *Global*
The whole architecture is involved in crisis management and the adaptation of its workflow.

Duration *Short*
Since the reconfiguration process is automated, a short duration can be expected (changing or creating new configurations).

Timeliness *Best-Effort*
Changes can coming during an adaptation process and must be faced.

Triggering *Time-trigger*
The evolution of the crisis is cached thanks to a background task requesting regularly for the state of the crisis. Information about functional characteristics of the firemen are collected thanks to sensors in a same way.

Effects. Effects on the Petals ESB are of administration order. Services are supposed to be stopped and (re-)started.

Criticality *Mission-critical*
An adaptation failure can cause critical risks and emergency stop should be planned for a human intervention.

Predictability *Deterministic*
Adaptation consequences on the system can be predictable.

Overhead *Reasonable*
The system is subjected to a low impact during workflow adaptation (stopping it, restarting it).

Resilience *Resilient*
The system must be as operational as before the change.

4.3.4 Adaptation

According to systems self-management definitions, this adaptation task is related to *self-configuring* and *self-protecting* [8].

Monitoring step consists in aggregating information from sensors located in the WSDM bus. They consist of non-functional and functional information about partners involved in the workflow.

Analyze step consists in detecting issues from aggregated information: crisis state evolution or limitations of a partner reached.

Plan step consists in conceiving a strategy according to the detected issue. For a crisis evolution, a reconfiguration workflow is deployed in order to add a branch on the functional workflow dealing with crisis management. For a detected deficient partner, an administrative mechanism is launched in order to stop some services and to replace them by operational ones.

Execution step controls the workflow execution, the administration invocation and tries to prioritize adaptations if several ones are requested at the same time.

4.3.5 Experimental Setup

The experimental setup will be initially done on dedicated testing servers at EBM Websourcing. It will consist in using a set of virtual servers in which Petals node(s) and WSDM monitoring node(s) will be installed.

4.4 Conclusion

This chapter illustrates SALTY framework and proposals in the context of an ESB middleware. Main features demonstrated thanks to these use cases are to tackle large-scale environments within a distributed architecture, to support adaptation of complex services orchestration by using technical and functional information. In addition of these features, we validate the use of MDE since adaptation framework will be built upon SCA models.

Bibliography

[1] Egee middleware architecture. `http://edms.cern.ch/document/476451`, July 2005.

[2] G. Avellino. Flexible job submission using web services: the glite wmproxy experience. (EGEE-PUB-2006-024), 2006.

[3] David Breitgand, Ealan Henis, and Onn Shehory. Automated and adaptive threshold setting: Enabling technology for autonomy and self-management. *Autonomic Computing, International Conference on*, 0:204–215, 2005.

[4] A Duarte, P Nyczyk, A Retico, and D Vicinanza. Monitoring the egee/wlcg grid services. *J. Phys.: Conf. Ser.*, 119:052014, 2008.

[5] E. Laure, F. Hemmer, F. Prelz, S. Beco, S. Fisher, M. Livny, L. Guy, M. Barroso, P. Buncic, Peter Z. Kunszt, A. Di Meglio, A. Aimar, A. Edlund, D. Groep, F. Pacini, M. Sgaravatto, and O. Mulmo. Middleware for the next generation grid infrastructure. (EGEE-PUB-2004-002), 2004.

[6] Diane Lingrand, Johan Montagnat, and Tristan Glatard. Modeling user submission strategies on production grids. In *International Symposium on High Performance Distributed Computing(HPDC'09)*, pages 121–130, June 2009.

[7] Mohammad Ahmad Munawar and Paul A. S. Ward. Leveraging many simple statistical models to adaptively monitor software systems. In *Parallel and Distributed Processing and Applications, 5th International Symposium, ISPA 2007, Niagara Falls, Canada, August 29-31, 2007, Proceedings*, volume 4742 of *Lecture Notes in Computer Science*, pages 457–470. Springer, 2007.

[8] Horn P. Autonomic Computing: IBM's perspective on the State of Information Technology. In *IBM corporation, http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf*, 2001.

[9] R. Sterritt, M. Parashar, H. Tianfield, and R. Unland. A Concise Introduction to Autonomic Computing. *Advanced Engineering Informatics*, 19(3):181–187, 2005.

https://salty.unice.fr/

ANR SALTY

Self-Adaptive very Large disTributed sYstems

Work Package:	WP1 - Requirements and Architecture
Coordinator:	UNS
Deliverable:	D-1.1 - Appendix C
Title:	**Internal truck tracking Scenario Specification**
Submission date:	2^{nd} August 2010
Project start date:	1^{st} November 2009, duration: 36 months
Revision:	209
Last change:	02.08.2010

Authors

(for this appendix)

Author	**Affiliation**	**Role**
J. Malenfant	UPMC	Lead
I. Truck	Univ. Paris 8	Lead
T. Bathias	Deveryware	Writer
O. Melekhova	UPMC	Writer
M.-A. Abchir	Deveryware & Univ. Paris 8	Writer
A. Pappa	Univ. Paris 8	Writer

Contents

CHAPTER 1

Introduction

1.1 Context

After the military for guidance and unit follow-up, geotracking popularized itself in several domains such as transportation and logistics. Recently, it is more and more adopted in the context of smartphones where numerous applications like route follow-up but also surrounding services (restaurants, hotels, ...) recommendation become widely available. Applications of geotracking currently explode in many areas: logistic, transportation, security, road traffic control, environmental tax collection on vehicles, etc.

In a nutshell, geotracking uses positioning information of mobile (persons, vehicle, ...) to follow them up in time and space so to use this information for application-specific purposes. Geotracking involves at least two entities: a positioning device and a tracking system. Positioning devices use different techniques to locate themselves. Under the global positioning system (GPS), devices triangulate their positions using signals from satellites. Mobile phones can be located from the geographic locus of the cell to which antenna it is currently connected. Finally, similar techniques can be used to locate WiFi cards from their wireless access points.

Positioning devices transmit positions to a tracking system. Most of the time, mobile phone networks are used for this purpose, but alternatives exist (*e.g.* satellite networks or WiFi). The tracking system is responsible for monitoring the position of mobiles (to which positioning devices are attached) and to trigger reactions when given conditions are met. Typical conditions are: being near, approaching or moving away from some point, approaching or moving away from another mobile, crossing a frontier (in the general sense), ...

Tracking systems can be directly embedded into end-user applications, such as a truck follow-up system, or can use dedicated platforms that correlate position information from several devices to trigger events sent to end-user applications. These platforms act as complex event processing (CEP) systems, but dedicated to geotracking. GeoHub is Deveryware geotracking platform, which not only correlates positions but also abstract end-user applications from the specifics of positioning devices and positioning techniques.

Geotracking faces two difficulties that will be addressed within the SALTY project. First, sending positions through a mobile phone network incurs a per message cost for customers which need to be minimized while keeping up with application requirements. Linked to this, and sometimes crucial, sending positions also requires energy from batteries which governs the autonomy of the device and so must also be minimized. Second, end-users lack the technical knowledge needed to configure the parameters of the geotracking, like the frequency of position reporting from devices, to catch up with application requirements, like notifying the arrival of a truck at a given location fifteen minutes in advance with a two-minute tolerance. Other points of interest concern fault-tolerance, *i.e.* coping with device malfunctions, and the overall workload of the geotracking platform which can't sustain more than a fixed number of position sendings from all of the connected devices to match its quality of service objectives in event processing (*e.g.* 50,000 positions per minute).

The cost minimization issue will be addressed by dynamically adapting the position reporting, first by modifying its frequency but also by switching back and forth from time-triggered to location-triggered geotracking when possible. As dynamic adaptation is the focus of the SALTY project, complementary adaptation *scenarii* will be provided by the applications themselves. The geotracking configuration, *i.e.* defining events to be notified to applications and the type of adaptations needed at run-time, will be addressed by developing an intelligent interactive configuration system.

1.2 Technical foundations

In this section, we review the major characteristics of the different system and devices used in the geotracking use case .

1.2.1 Positioning devices

Geotracking relies on positioning devices which characteristics deeply impact costs and resource usage. Positioning devices first fall into three categories of position means:

1. Global Positioning System (GPS) devices are now well-known; they use information and triangulation of signals coming from satellite constellations. Their precision highly depends upon the number of satellites they can ”see”, the angle between them and even the ground configuration which can cause some bias. After computing positions, devices typically push them to a server using data connection over a GSM network.

2. GSM cell-id uses readily available phone (or similar devices connected to that network) tracking information, *i.e.* the cell in which the device actually is, to provide estimated positions. Its precision highly depends upon the density of antennas and the geographical form of the cell in which the phone is. It needs no specific intervention from the phone itself but is rather a service offered by the cell phone operator network infrastructure. In summary, upon a call to their API, operators provide estimated positions from the geographical locus of the cell containing the phone.

3. WiFi cell-id is similar to GSM cell-id but rather uses WiFi antennas to estimate positions of devices connected to that network. Again, the precision of the method depends upon the density of the antennas.

Devices can be fixed or portable. Indeed, specific GPS devices can be attached to vehicles and be powered by these. Such devices are usually also connected to sensors on the vehicle (door open/closed, motor temperature and speed, fuel level, ...) and can send sensor data along with positions. But more and more portable GPS devices are used for their flexibility, at the expense of running on batteries, which then become a scarce resource. Mobile phones share characteristics with portable GPS, while WiFi cell-id is totally dependent on the device to which the WiFi card is connected.

Most GPS devices send positioning (and possibly other data) through a data link over the GSM network at a frequency that can be dynamically modified by sending them an appropriate command. GPS devices, for example, always send their current speed estimated from their successive positions.

More and more programmable devices appear everyday on the market. Such devices can embed application specific code that can drive the positioning but also adaptations

of this process to some extent. Such devices can also group positioning and other monitoring data to send them at a less regular frequency. We will therefore distinguish:

measurement frequency: the frequency at which positioning and other monitoring data are *measured on the vehicle*;

transmission frequency: the frequency at which positioning and other monitoring data are *transmitted to the geotracking system*.

Geotraking hence involves a decision over these frequencies, decisions that are currently made once and for all, but that we will strive to make and adapt at run-time. The simplest positioning devices are limited to this end. A variant fosters the use of programmable devices, capable of deciding on their own when to measure and transmit data when equipped with appropriate programs. Programmable devices can either be programmed in advance or dynamically uploaded with new programs.

The standard behavior of positioning devices in geotracking is to send positions regularly, at a given frequency. We will refer to this behavior as *time-triggered geotracking*. However, the flexibility of programmable devices can be leveraged to adopt novel, non-standard approaches to trigger data transmission. In some cases, devices can be instructed to send positions when certain spatial conditions are triggered, like being near some known location. We will refer to this behavior as *location-triggered geotracking*. Geotracking will therefore require a decision between two data-transmission modes:

time-triggered mode, where data is transmitted based upon a time condition (frequency, delay or absolute time), and

location-triggered mode, where data is transmitted upon a location condition (crossing a frontier, approaching some point, etc.).

As the most prominent mean to transmit the data is the GSM network, devices are also equipped with a communication subsystem similar to the one of mobile phones. Hence, positioning devices using the GPS method can also be located using the cell-id positioning method from the GSM network. This alternative can be used to get more robustness in the geotracking.

Energy consumption is an important aspect for all mobile positioning devices operating on batteries. Taking position measurements and transmitting data incur energy costs, as well as maintaining the link to the GSM antennas. Some positioning devices can manage their power consumption by selectively powering on and off subsystems such as the position measurement and the communication subsystems. The device can also put itself in sleep mode where the wake-up can be planned at a specific time or in reaction to some event. As the power-up incurs some cost in energy and time delay, a trade-off must be made when shutting down subsystems or going to sleep mode, as the energy consumption of the power-up must be amortized by the one saved during the power-off or sleep.

Devices power management and particularly selectively powering down the GSM subsystem means that it can be impossible to send them commands at some time.

1.2.2 Deveryware GeoHub

Deveryware GeoHub is the geotracking platform used throughout the experiments. GeoHub is a middleware that can locate mobiles, compute information based upon received locations and send them to applications using a web service API. Mobiles can be located

using a combination of technologies: GPS, wifi, GSM cell-id via network, via mobile or hybrid. Several network links can be used to send informations: GSM/SMS or GPRS, wifi, satellites. Computation include storing data, computing reports, displaying maps, generating alerts (see below). Applications can use all GeoHub functions using XML-RPC web services with ten's of functions.

For SALTY, the GeoHub serves essentially two purposes:

- abstract applications away from particular positioning devices and methods by offering them standard positioning interfaces,
- efficiently correlate position information from several positioning devices to detect more complex events to be notified to applications.

As mediator between applications and positioning devices, GeoHub offers a unified interface to send commands to devices. Commands are sent using either GPRS or SMS. Communication latencies in the order of 2 to 3 seconds must be expected from GPRS, but they reach in the order of one minute for SMS; they must be taken into account in the decision process.

GeoHub alert service is a mechanism allowing an application to be notified of particular events. All received data can be used to generate events. This mechanism is used through the GeoHub API. The following elements are used:

- mobile id to be monitored,
- active date/time to monitor,
- script used for monitoring,
- notification to send when an event is detected.

The mobile id specifies the mobile to monitor. The alert will be active within the time frame specified for its activation. It can be started immediately if necessary (In this case a delay of 2 or 3 seconds can be necessary for the alert to become active). An alert can, for example, be specified active from now until tomorrow, or only on Mondays from 1pm to 5pm. In any case, the GeoHub will send a specific message to the application at each beginning of a monitoring period.

The triggering of an alert itself is described by a program that will be executed each time new data are received from this mobile id. Currently, such programs are written in Forth and, to guarantee the quality of service for all users, get allocated a limit of a thousand instructions to be executed each time they are launched by GeoHub. This program will be provided with the last data just received and the data used at its previous call (to be able to compare them). The program will also be executed regularly even if no new data is received. The program have to decide if it need to throw a alert. In this case, the GeoHub will take care of executing actions attached to this alert: send a message to the application and send notification to users (SMS, e-mail, fax, phone call with speech synthesis).

GeoHub also keeps track of other monitoring data sent by the devices, store them in its own database and provide applications with means to query the database or download them.

1.2.3 Geotracked vehicle and their operation

Positioning devices on geotracked vehicle are operated by drivers. At departure, drivers can notify the start of the geotracking through the device using a trigger available on most devices. At arrival, they can notify the corresponding end of the geotracking. Each time they stop voluntarily for some reason (having some rest, eating, ...), drivers can also notify the stop and restart through the device. This information can then be used by the geotracking system to interpret positioning data and react accordingly: unnotified stops can be interpreted as accidents or traffic jams needing intervention or route re-computation.

CHAPTER 2

Scenario 1 - Long distance truck tracking

The first scenario appears in the context of the critical economic area of logistic. Long distance truck tracking is concerned with following up trucks which deliver goods from warehouses to warehouses in a complex network of logistic bases. This scenario is typical either for general in transportation and large-scale distribution.

2.1 Context

2.1.1 Overall goals of the supporting application

Logistics or distribution companies need to track their truck fleets so to optimize the use of their logistic bases, to minimize the travel time of their trucks, and to minimize their waiting time when they arrive at logistic bases to download and upload goods.

Coarsely speaking, the application is assumed to be made of four different parts:

- devices attached to vehicles that can at least push data at some updatable frequency, but some may also be able to have software executing on them that is updatable at runtime;
- GeoHub used to host and execute triggering code for notifications;
- the application itself, planning the routes, tracking each truck individually, and reacting to the notifications relating to them;
- the adaptation module, based on autonomic computing principles, that will receive the necessary information required to make decisions and trigger adaptations upon all of the other modules.

2.1.2 Long distance tracking use cases

Tracking long distance transportation trucks is founded on the following hypothesis:

- Truck routes are planned in advance, as well as their intermediate and final destinations. The objective of the geotracking is to make sure that trucks follow their route, pass by the different intermediate destinations as well as other predefined waypoints, and reach their final destination. Trucks can deviate from their route to some extent, but if the deviation becomes too large, the reason for this deviation must be established and corrective actions taken, such as ordering the driver to come back to the planned route or replan the route when necessary (accidents, traffic jams, unplanned road construction, ...).
- Drivers of tracked vehicles always notify their departure, arrival, stop and restart using appropriate commands of their positioning devices.

- Geotracking lasts from their departure until their final destination.
- The main positioning device for some trucks will have a fallback, namely the mobile phone of the driver located under cell-id. In some failure scenario, the main positioning device may itself be its own backup: if only the measurement unit is faulty, the mobile network transmission subsystem can still be located under cell-id.

This scenario involves three major geotracking objectives:

1. **Notification of arrival at intermediate destinations**, where it is required that trucks arriving at a warehouse notify their arrival enough in advance to let the warehouse coordinator allocate them a port so to optimize port usage and the waiting time of trucks. After notification of their arrival, trucks approaching warehouses are still closely tracked in order to take corrective actions if it appears that they will be too late or way in advance.
2. **Imposed corridor**, where trucks are forced to stay within a corridor around their route to make sure they don't deviate much of this route. When trucks deviate from their corridor, corrective actions are taken, first to inquire drivers from the reason of the deviation, and if justified, replanning the rest of their route. The system will get inputs from traffic control and weather services so that it may confirm or detect itself conditions justifying a replanning of the route.
3. **Waypoint notification**, where the passage nearby some predefined points must be notified. As corridor enforcement deals with deviation from the planned route, waypoint notifications are used to keep track of the progress of the truck, for example by displaying passed waypoints along with their passage time on a map.

Each of the technical parameters and adaptations required for these geotracking objectives are detailed in the following sub-*scenarii*.

2.1.3 Overall description of the adaptation scenarii

The previous business-oriented objectives of the long distance truck tracking application can only be achieved if the positioning of trucks is precise and timely enough to match the desired degree of precision in the different notifications. In traditional geotracking applications, some statically determined minimal frequencies of measurement and transmission of positions are imposed to devices. The devices are then set to that frequency, that never changes during the application execution. Such approaches incur a high cost in position transmission, as the frequency set to get the necessary precision on the most demanding notifications is used all the time, even when the mobile will not approach critical points before hours. A smarter use of geotracking is to keep the rate of measurement and transmission as low as possible when no critical point is approaching, but to progressively augment the frequency when necessary. Another closely related adaptation is the switch back and forth between the time-triggered and the location-triggered modes.

Precision in the positioning can also be adjusted by choosing among several positioning methods the one with the lowest cost for the required precision at any time. Also, when devices are partly malfunctioning, such as having no position or aberrant ones, a decision can be made to switch to an alternative positioning method. Finally, to lower the energy consumption, decisions can be made to power on and off subsystems, when possible.

These adaptations are discussed more thoroughly in the next sections, for each geotracking objectives of the application.

2.2 Overall experimental setup

Testing and experimenting applications that have at least part of their behavior bound to real time is inherently difficult. In the case of truck tracking, the real time behavior comes from the need for real positions sometimes correlated with real geographical positions and artifacts, such as roads and their speed limits, warehouse positions, waypoints, etc.

Hence, it will not suffice to merely generate input data and run an application on a standard computer. The experiment will have to take place in the real time frame, positions and transit times being realistic to some extent. On the other hand, as scalability to large numbers of geotracked vehicles is a key issue in the SALTY project, it is not realistic to get thousands of real trucks going on the road for the experiments; they will require simulated vehicles. Simulated vehicles are pieces of software executing on a computer and sending positions and receiving commands from the GeoHub, as real vehicles do.

The GeoHub, the application and the adaptation layer used in the experimentation will be the real ones[1]. But trucks and their positioning devices will be simulated as threads on stock computers, executing simulated travels. The idea is to get a planned route, using planning services such as ViaMichelin, and then plan a per truck simulation scenario to be run around that route. Each truck simulation will exhibit required notifications, and therefore adaptation *scenarii*. In some cases, incidents, like device malfunctions, traffic jams, etc. will be injected into the *scenarii* to trigger the corresponding adaptations.

As a large number of such truck simulation *scenarii* will be required, they will be generated automatically from the following sub-*scenarii* and their variants. A machine-readable description of these will be constructed, and fed into a simulation *scenarii* generator that will give an executable *scenario* in another machine-readable format used by specifically developed truck and positioning device simulation programs. To our knowledge, no such simulation platform exists to date. It will be a contribution of the SALTY project that could be reused by Deveryware in the future to test other enhancements of their platform.

2.3 Sub-scenario 1.1 - Notification of arrival at intermediate destinations

2.3.1 Objective

When trucks approach logistic bases, coordinators of these bases need to be notified so to schedule the truck to a port. Ports are scarce resources, so coordinators need to maximize their use time, but not at the expense of wasting the time of trucks and drivers waiting for downloading or uploading.

Parameters: the notification delay and the tolerance in this delay, a time limit at which the bases must be notified if the vehicle is to arrive in advance and the tolerance in this limit, a time limit at which the bases must be notified if the vehicle is to arrive late and he tolerance in this limit. The figure 2.1 shows the different limits towards the destination.

[1] A copy, for the GeoHub, to avoid service denial to actual custumers of Deveryware.

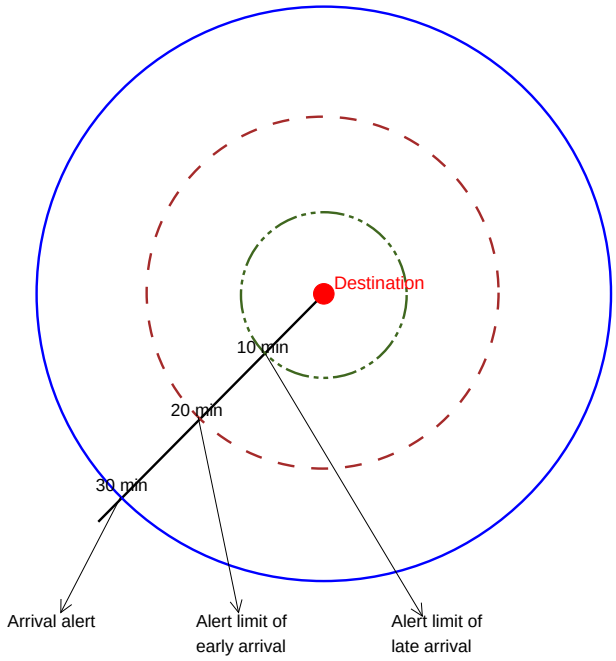


Figure 2.1: Arrival at intermediate destinations.

Typical values: 30-minute arrival, 20-minute early arrival and 10-minute late-arrival notifications, each with a two-minute tolerance.

This sub-scenario can develop into several variants, depending upon the exact behavior of the truck approaching the destination. We plan to test particularly the following variants. In each of them, the truck is assumed to have notified its arrival correctly at the estimated 30-minute to arrival point. Variants look at different possibilities for the early arrival and late arrival time limits:

1. The nominal variant, where the truck passes the early arrival and late arrival time limits while being just in time. Positions of the truck will show this, and the application will simply need to notify the arrival at the warehouse.

2. The late arrival variant, where the sequence of positions of the truck will show its predictable late arrival and the truck will arrive late. The application will be notified of the arrival and then of the late arrival.

3. The early arrival variant, where the sequence of positions of the truck will show its predictable early arrival and the truck will arrive in advance. The application will notify the arrival and then the early arrival.

4. The erroneous position variant, where the positions sent by the positioning device become unreliable which is detected by the application. In this case, the application will notify the arrival, but no early nor late arrivals; it will rather notify (internally) an erroneous positions exception. The detection of erroneous positions is left to the application, which can use several techniques to do so, such as comparing the position to the previous ones and consider them erroneous if the estimated time to reach the new position from the previous one exceeds the capability of the truck.

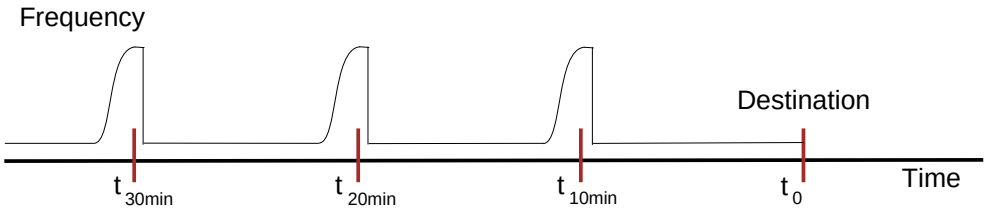


Figure 2.2: Expected modifications in the transmission frequency for the intermediate destinations arrival notification sub-scenario.

5. The network failure variant, where there is a loss of connection with the device. In this case again, the application will notify the arrival, but no early nor late arrivals; it will rather notify (internally) a network failure exception.

6. The device failure variant, where the device becomes faulty. In this case again, the application will notify the arrival, but no early nor late arrivals; it will rather notify (internally) a device failure exception.

7. The lost truck variant, where the application stops receiving data and the truck never arrives at destination. In this case again, the application will notify the arrival, but no early nor late arrivals; it will rather notify (internally) a lost truck exception, which in turn will be treated by the application by issuing an application-dependent search command.

Each of these variants will lead to a complete behavioral scenario for at least one truck in the experiments. Expected adaptations that will be triggered during the geotracking of this truck are presented in the next subsection.

2.3.2 Classification

Below we provide an analysis of the described adaptation in the context of the modeling dimensions from appendix A:

Goals. Minimize the overall costs of position sendings, minimize the probability of missing notifications, minimize the energy consumption of the device.

Evolution: *Static*
In this sub-scenario, goals do not change within the lifetime of the system.

Flexibility *Rigid*
The goals are prescriptive.

Duration *Persistent*
The goal is valid throughout the system's lifetime. However, as the truck passes from intermediate destinations to other intermediate destinations and to its final destination, the arrival position to be considered passes from one to the next.

Multiplicity *Multiple goals*
In this sub-scenario, we consider three different goals.

Dependency *Dependent*
The three goals are conflicting, as higher frequencies minimizes the probability of missing notifications, but cost more and consume more energy. Trade-offs between them must be sought.

Change. The adaptation concerns the frequency of position measurement and transmission, the positioning mode and the sleep mode of the device. It is mainly triggered when the position of the truck is approaching the destination point and its different limits. Other conditions that influence the adaptation are: the speed of the truck, the road conditions, the battery level of the positioning device.

Source *External and internal*
The main source of changes is external, the position of the truck. Others are internal.

Type *Functional and non-functional*
Positions are functional, but other sources are non-functional.

Frequency *Frequent*
As trucks approach destinations and the previous notification limits, the changes will be made to gradually increase the frequencies.

Anticipation *Unforeseeable (mostly)*
Positions of trucks are hardly foreseeable, as speed depends upon not only on the type of road but also from road, traffic, and weather conditions. Hence, adaptation will be triggered by positions transmitted by the truck.

Mechanism. Changes concern parameters of the positioning devices: which device to use, frequencies, mode, ...

Type *Parametric*
Changes concern parameters of the positioning devices: which device to use, frequencies, mode, ...

Autonomy *Autonomous*
No outside intervention.

Organization *Decentralized*
The adaptation is performed by the autonomic manager of the arrival management component, but has its effect on the positioning device of the truck, so the latency in the adaptation must be taken into account.

Scope *Local*
Adaptation only involves changes in the positioning device.

Duration *Short to very short*
The amount of time required to reconfigure the device should be from seconds to minutes. Reconfiguration have no impact on the availability of the system.

Timeliness *Guaranteed*
A time-bound must be observed for this kind of feedback control to be efficient.

Triggering *Event-trigger*
The main triggering conditions are event-based, by having positions matching certain conditions or by matching certain conditions at some time.

Effects. Geotracking objectives are matched but with better resource usage and lower cost.

Criticality *Mission-critical*
Although geotracking is currently done without adaptation of frequencies or fault-tolerance mechanisms, no guarantee can be put that no notification will

be missed for any given precision. To have such guarantee, to some limit, adaptation is mission-critical.

Predictability *Probabilistic*
The result of actions in terms of system guarantees can only be measured as higher probabilities to match the objectives.

Overhead *Insignificant*
Parameters reconfiguration takes a small amount of time.

Resilience *Resilient*
Devices can be adapted without stopping, and the adaptation does not impair their operation, as the time to adapt is small compared to the typical time intervals between positioning instants.

2.3.3 Adaptation

In a nutshell, this sub-scenario is concerned with the need to get a position of the truck within a location interval which bounds depends upon the given notification tolerance. Time limits must first be translated into locations on the given route, a translation which depends itself upon the speed and the nature of the route. The different bounds can be computed during this translation. As devices send positions, the problem is to make sure to get a position when the truck will be in the interval. From the speed of the truck within the route segment where the interval is, one can compute the frequency at which positions must be taken and transmitted to ensure that one will be in the interval. As the target frequency may be high, and anyway unnecessary when the truck is still far away from the destination, the basic control to be applied is to use a low frequency as long as the truck is far away but to raise it to the computed value when it will approach the notification interval. The figure 2.2 shows how the frequency is expected to evolve. The frequency is low by default, but when approaching the different intervals it shall gradually raise to the required level and then switched back to default after the notification is made.

Expected adaptations for each scenario variant presented in the preceding subsection are as follows:

1. In variant 1, as shown in the left part of Figure 2.2, a progressive increase in the transmission frequency is expected to go from the default frequency to the (higher) frequency that ensures a position transmission within the location interval corresponding to the notification delay and its tolerance. The frequency shall come back to default after the notification of the arrival.

2. In variant 2, as shown in the right part of Figure 2.2, after the notification of the arrival, a similar increase-decrease of the frequency is expected to notify correctly the late-arrival. If the late-arrival is due to traffic jams, an application-specific adaptation may call for a re-computation of the route to destination.

3. In variant 3, as shown in the middle part of Figure 2.2, after the notification of the arrival, a similar increase-decrease of the frequency is expected to notify correctly the early-arrival.

4. In variant 4, after the adaptation for the arrival notification, the frequency may increase again if the positions mislead the application to observe an early or late-arrival but as soon as the positions will be determined as erroneous, the application

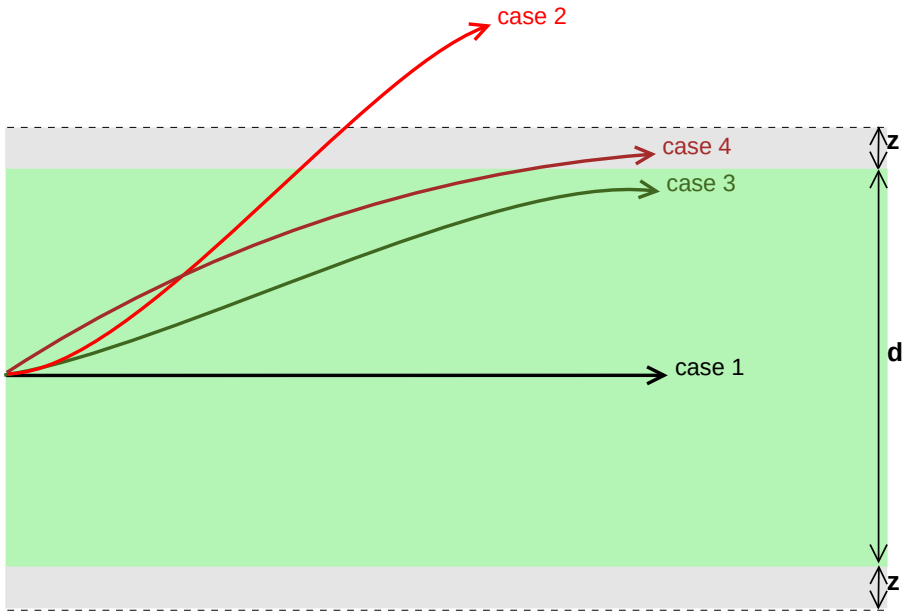


Figure 2.3: Main alternative deviations from the planned route.

will inhibit any other adaptation, notify an erroneous position exception, which will itself trigger a decision to switch to the fallback positioning devices if any.

5. In the variant 5, after the adaptation for the arrival notification, if an alternative positioning device using another network for transmission is available, a switch to this device will be triggered.

6. In variant 6, after the adaptation for the arrival notification, if an alternative positioning device is available, a switch to this device will be triggered.

7. In variant 7, the adaptation for the arrival notification will appear, but no other one.

2.4 Scenario 1.2 - Imposed corridor

2.4.1 Objective

Trucks follow routes from which deviations must be notified to their fleet manager. Deviations resulting from road construction or repair, or from traffic jams should trigger the computation of an alternative route, but unjustified deviations requires immediate intervention especially when highly valuable goods are involved. To detect deviation, a corridor is allocated around the route, within which the truck must remain. The figure 2.3 shows the four major cases that can happen: (1) the vehicle follows its nominal route, (2) the vehicle clearly crosses the corridor, (3) the vehicle approaches the frontier of the corridor but while it remains inside it, it requires a finer-grain geotracking, and (4) the vehicle crosses the frontier but within a tolerated distance, hence notification will be delayed until some time limit if the vehicle does not come back in the corridor.

Parameters: the route to follow, the width of the allocated corridor, the tolerance in distance from the corridor frontier to detect the crossing, as well as a time tolerance

that limits the time the truck can stay outside the corridor but within the distance tolerance.

Typical values: a route obtained from a mapping service like ViaMichelin, a corridor spanning 5 kilometers on both sides of this route, a 500-meter tolerance to detect the crossing and a limit of 10 minutes outside the corridor but within 500 meters tolerance before notifying the crossing anyway.

Again, this sub-scenario can develop into several variants, depending upon the exact behavior of the truck approaching the destination. We plan to test particularly the following variants:

1. The nominal variant, where the truck follows the corridor without approaching the limits of the corridor at a distance less than the tolerance. No notification is done to the application. This variant corresponds to case 1 in Figure 2.3.

2. The admitted corridor crossing variant, where the truck crosses the limit of the corridor because of an accident or a traffic jam on the main route and begins to transmit positions outside the tolerance limit. In this case, the application notifies this crossing, but after verification will trigger a re-computation of the route. This variant corresponds to case 2 in Figure 2.3.

3. The rejected corridor crossing variant, where the truck crosses the limit of the corridor for no good reasons (maybe for bad reasons, like stealing the content of the truck or passing in another country) and begins to transmit positions outside the tolerance limit. In this case, the application notifies this crossing and issues a search and bring back request. This variant also corresponds to case 2 in Figure 2.3.

4. The near-crossing but inside variant, where the truck approaches the limit of the corridor, but stays inside it. No notification should be issued in this case. This variant corresponds to case 3 in Figure 2.3.

5. The near-crossing but outside variant, where the truck crosses the limit of the corridor, but stays within the distance tolerance. In this case, a notification will be issued if the trucks stays in this zone for a time longer than the time limit without going back to the corridor. This variant corresponds to case 4 in Figure 2.3.

2.4.2 Classification

Below we provide an analysis of the described adaptation in the context of the modeling dimensions from appendix A:

Goals. Minimize the overall costs of position sendings, minimize the probability of missing notifications, minimize the energy consumption of the device.

Evolution: *Static*
In this sub-scenario, goals do not change within the lifetime of the system.

Flexibility *Rigid*
The goals are prescriptive.

Duration *Persistent*
The goal is valid throughout the system's lifetime.

Multiplicity *Multiple goals*
In this sub-scenario, we consider three different goals.

Dependency *Dependent*
The three goals are conflicting, as higher frequencies minimizes the probability of missing notifications, but cost more and consume more energy. Trade-offs between them must be sought.

Change. The adaptation concerns the frequency of position measurement and transmission, the positioning mode and the sleep mode of the device. It is mainly triggered when the position of the truck is approaching the frontier of the corridor. Other conditions that influence the adaptation are: the speed of the truck, the road conditions, the battery level of the positioning device.

Source *External and internal*
The main source of changes is external, the position of the truck. Others are internal.

Type *Functional and non-functional*
Positions are functional, but other sources are non-functional.

Frequency *Frequent*
As trucks approach frontiers of the corridor, the changes will be made to gradually increase the frequencies.

Anticipation *Unforeseeable (mostly)*
Positions of trucks are hardly foreseeable, as speed depends upon not only on the type of road but also from road, traffic, and weather conditions. Hence, adaptation will be triggered by positions transmitted by the truck.

Mechanism. Changes concern parameters of the positioning devices: which device to use, frequencies, mode, ...

Type *Parametric*
Changes concern parameters of the positioning devices: which device to use, frequencies, mode, ...

Autonomy *Autonomous*
No outside intervention.

Organization *Decentralized*
The adaptation is performed by the autonomic manager of the corridor management component, but has its effect on the positioning device of the truck, so the latency in the adaptation must be taken into account.

Scope *Local*
Adaptation only involves changes in the positioning device.

Duration *Short*
The amount of time required to reconfigure the device should be from seconds to minutes. Reconfiguration have no impact on the availability of the system.

Timeliness *Guaranteed*
A time-bound must be observed for this kind of feedback control to be efficient.

Triggering *Event-trigger*
The main triggering conditions are event-based, by having positions matching certain conditions or by matching certain conditions at some time.

Effects. Geotracking objectives are matched but with better resource usage and lower cost.

Criticality *Mission-critical*
Although geotracking is currently done without adaptation of frequencies or fault-tolerance mechanisms, no guarantee can be put that no notification will be missed for any given precision. To have such guarantee, to some limit, adaptation is mission-critical.

Predictability *Probabilistic*
The result of actions in terms of system guarantees can only be measured as higher probabilities to match the objectives.

Overhead *Insignificant*
Parameters reconfiguration takes a small amount of time.

Resilience *Resilient*
Devices can be adapted without stopping, and the adaptation does not impair their operation, as the time to adapt is small compared to the typical time intervals between positioning instants.

2.4.3 Adaptation

In this sub-scenario, expected adaptations are the following:

1. In the variant 1, no adaptation is expected.
2. In variant 2, an increase/decrease schema in the transmission frequency, similar to the one shown in the arrival notification sub-scenario, is expected when the truck will approach and then cross the limit of the corridor and its tolerance. After the re-computation of a new route, though, the route and its corridor will be adapted and then no further notification of being outside the (original) corridor shall be issued.
3. In variant 3, an increase/decrease schema in the transmission frequency is also expected, and after the crossing an emergency follow-up frequency will be adopted at which every new position of the truck will be notified to the application automatically.
4. In variant 4, an increase in the frequency is expected as the truck approaches and stays within the tolerance zone inside the corridor. The frequency should remain high as long as the truck stays in this zone, but decrease when it comes back in the normal zone of the corridor.
5. The variant 5 is similar to variant 4, but ending with a notification and an emergency follow-up as in variant 3.

2.5 Scenario 1.3 - Waypoint notification

2.5.1 Objective

Long distance trucks often follow very long routes that can span over entire continents. Fleet coordinators need to be kept informed of the progress of the truck along this route by setting waypoints which crossing shall trigger a notification. An alternative usage

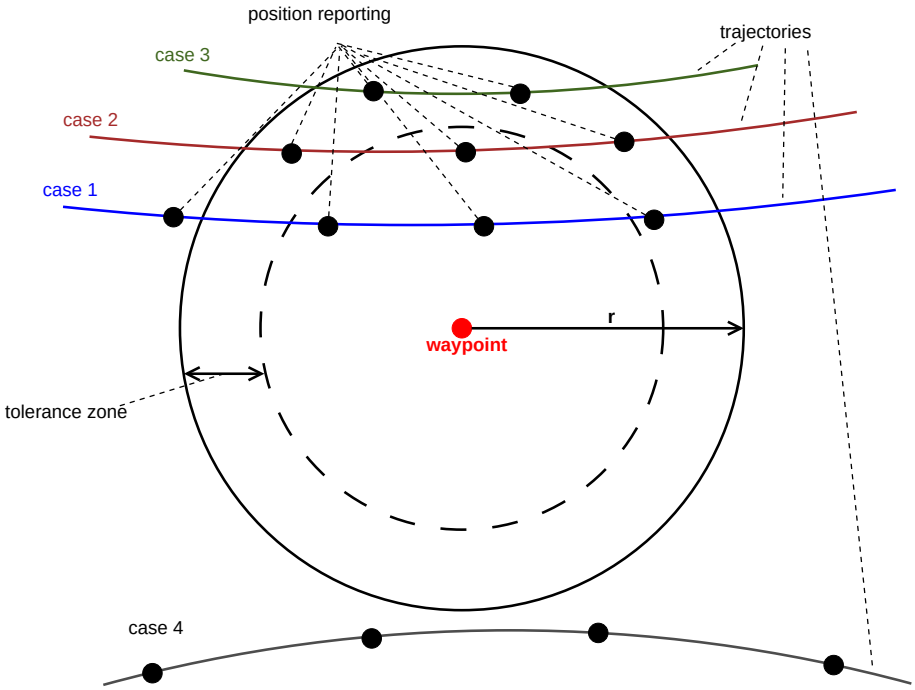


Figure 2.4: Different trajectories against waypoints.

of waypoints is to make them virtual tollbooth to collect environmental taxes (on a pay per passage basis). When waypoints are used as tollbooth, a single position may not be sufficient to prove that the vehicle passed the toll; more stringent requirements, such as having multiple positions within a radius from the point, may be imposed. The figure 2.4 shows these different cases.

Parameters: the position of the waypoints, the radius of the notification area, a tolerance on this radius and a triggering rule.

Typical values: waypoints as known positions at roads crossing, a 200-meter radius with a tolerance of 25 meters, the notification being triggered if three positions can be reported within the radius, from which at least two are not in the tolerance zone.

Many potential problems are raised by waypoint notifications. One of them is the possibility of passing over the point a second time on some road configurations, like highway exits. When this second passage happens very rapidly after a first passage, it should not trigger a second notification. Although such cases should be avoided, putting waypoints at highway crossings might be a clever choice (especially for toll purposes), hence requiring to deal with this problem. One possibility is to deactivate the waypoint for a given vehicle for some time after a passage.

This sub-scenario develops around the following variants:

1. The nominal variant, where the trucks crosses the waypoint area with at least two positions inside the area over three in the area plus its tolerance zone. The application must be notified to the passage. This variant corresponds to case 1 in Figure 2.4.

2. The near-miss variant, where the truck has one position inside the waypoint area and maybe others in the tolerance zone. The application must be notified to the near-miss. This variant corresponds to case 2 in Figure 2.4.

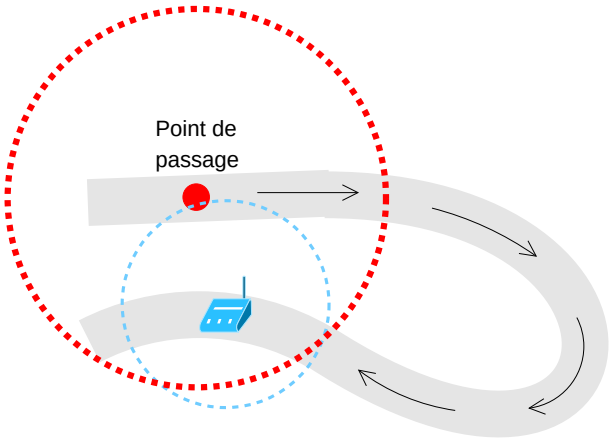


Figure 2.5: Potential second passage on a waypoint.

3. The miss variant, where the truck has at least one position inside the waypoint tolerance zone but none inside. The application must be notified to the miss. This variant corresponds to case 3 in Figure 2.4.

4. The no-passage variant, where no position appear inside the waypoint area nor its tolerance zone. No notification should occur. This variant corresponds to case 4 in Figure 2.4.

2.5.2 Classification

Below we provide an analysis of the described adaptation in the context of the modeling dimensions from appendix A:

Goals. Minimize the overall costs of position sendings, minimize the probability of missing notifications, minimize the energy consumption of the device.

Evolution: *Static*
In this sub-scenario, goals do not change within the lifetime of the system.

Flexibility *Rigid*
The goals are prescriptive.

Duration *Persistent*
The goal is valid throughout the system's lifetime. However, as the truck passes from waypoints to waypoints, the waypoint to be considered passes from one to the next.

Multiplicity *Multiple goals*
In this sub-scenario, we consider three different goals.

Dependency *Dependent*
The three goals are conflicting, as higher frequencies minimizes the probability of missing notifications, but cost more and consume more energy. Trade-offs between them must be sought.

Change. The adaptation concerns the frequency of position measurement and transmission, the positioning mode and the sleep mode of the device. It is mainly triggered when the position of the truck is approaching the waypoint. Other conditions that

influence the adaptation are: the speed of the truck, the road conditions, the battery level of the positioning device.

Source *External and internal*
The main source of changes is external, the position of the truck. Others are internal.

Type *Functional and non-functional*
Positions are functional, but other sources are non-functional.

Frequency *Frequent*
As trucks approach waypoints, the changes will be made to gradually increase the frequencies.

Anticipation *Unforeseeable (mostly)*
Positions of trucks are hardly foreseeable, as speed depends upon not only on the type of road but also from road, traffic, and weather conditions. Hence, adaptation will be triggered by positions transmitted by the truck.

Mechanism. Changes concern parameters of the positioning devices: which device to use, frequencies, mode, ...

Type *Parametric*
Changes concern parameters of the positioning devices: which device to use, frequencies, mode, ...

Autonomy *Autonomous*
No outside intervention.

Organization *Decentralized*
The adaptation is performed by the autonomic manager of the waypoint management component, but has its effect on the positioning device of the truck, so the latency in the adaptation must be taken into account.

Scope *Local*
Adaptation only involves changes in the positioning device.

Duration *Short*
The amount of time required to reconfigure the device should be seconds to minutes. Reconfiguration have no impact on the availability of the system.

Timeliness *Guaranteed*
A time-bound must be observed for this kind of feedback control to be efficient.

Triggering *Event-trigger*
The main triggering conditions are event-based, by having positions matching certain conditions or by matching certain conditions at some time.

Effects. Geotracking objectives are matched but with better resource usage and lower cost.

Criticality *Mission-critical*
Although geotracking is currently done without adaptation of frequencies or fault-tolerance mechanisms, no guarantee can be put that no notification will be missed for any given precision. To have such guarantee, to some limit, adaptation is mission-critical.

Predictability *Probabilistic*
The result of actions in terms of system guarantees can only be measured as higher probabilities to match the objectives.

Overhead *Insignificant*
Parameters reconfiguration takes a small amount of time.

Resilience *Resilient*
Devices can be adapted without stopping, and the adaptation does not impair their operation, as the time to adapt is small compared to the typical time intervals between positioning instants.

2.5.3 Adaptation

In this sub-scenario, expected adaptations are the following. In the three first variants, the intersection of the road with the waypoint area gives the segment of the route within which positions must be obtained. Hence, we face two problems: detecting when the truck will enter the waypoint area and the duration of this passage to compute the frequency of measurements to get the positions. The adaptation consists of increasing the frequency of transmission to capture the entrance in the waypoint area, and then to set the frequency of measurements prior the passage at the required level to get the positions. In this sub-scenario, we postulate that positions in the waypoint can be transmitted at the end of the passage.

In the last variant, an increase in the transmission frequency will happen if the passage is near the limits of the waypoint area, but no change in the measurement frequency as no notification will be needed.

CHAPTER 3

Scenario 2 - Short distance truck tracking

3.1 Context

3.1.1 Overall goals of the supporting application

The short distance truck tracking scenario deals with fast delivery parcel services where delivery men visit customers either to deliver or pick up parcels. Such companies need to track their truck in order to optimize tour delivery and add locations where to pick up parcels during the round when time permits. An important criticism towards such companies is the long delivery time window they impose to their customers; minimizing this is therefore an important business objective.

Geotracking can help to shorten these windows by following more closely the progress of trucks in order to notify customers when the delivery will be late, and replan the route to deliver most customers on time even by skipping some when necessary. Assume customers must be delivered one after the other, each within a time window announced in advance. No later than a certain deadline before the end of each time window, the corresponding customer must be notified if the delivery appears to be late. If a customer cannot be delivered on time, the route coordinator may decide to skip come back later to this customer in order for the delivery man to serve the rest of the customers on time. The goal is to maximize the number of customers delivered on time, and therefore their satisfaction, even at the expense of a very late delivery to some of them.

3.1.2 Short distance tracking use cases

Short distance tracking scenario is based on the following hypothesis :

- Rounds are planned in advance and customer announced a fixed delivery window that must be matched by the delivery man.
- Delivery men always notify their departure, arrival, stop and restart using the appropriate command on their positioning device.
- The geotracking will last from departure to the return at the logistic base.
- The main positioning device for some trucks will have a fallback: the mobile phone of the driver located under GSM cell-id. In some failure scenario, the main positioning device may itself be its own backup it the measurement unit is faulty but the mobile network transmission subsystem can still be located under GSM cell-id.

The scenario involves the following geotracking objectives:

1. **Notification of late arrival**, where trucks must deliver within an time window (with some tolerance) several destinations. When the truck positions and its route show

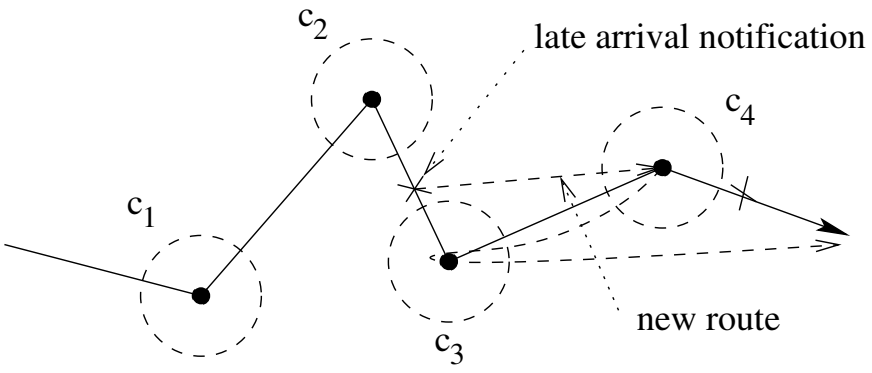


Figure 3.1: Route for the delivery and collecting of parcels.

that the delivery of a customer cannot be on time, a notification must be sent to the customer no later than a certain notification delay before the end of his delivery time frame. If the delivery is forecasted to be late, a maximum delay is set, such that when the customer cannot be delivered within this delay, it will be rescheduled later in the day, using route replanning to get a new round for the delivery man.

2. **Imposed corridor**, where trucks are forced to stay inside a certain corridor around their route, otherwise a notification must be sent to the route coordinator. If a route replanning is required, a new corridor will be imposed according to the new route. This objective is essentially the same as in the long distance truck tracking use case, so it will not be detailed again here.

3.1.3 Overall description of the adaptation scenarii

As in the long distance use case, the notification of late arrival and corridor enforcement cannot be achieved unless the location of trucks is precise and timely enough. However, precision is only needed when trucks approach the time limit for late arrival notification to the next customer, or when they approach the frontiers of their corridor. Hence, the main adaptation aims at ensuring that positions are pushed without failure and at the right frequency when these conditions are to be met.

Unlike the long distance use case though, the route replanning involves *a priori* a human decision from the route coordinator. In practice, this is more likely to happen than a purely automatic adaptation that may decide to skip an important customer. Techniques can be used to automatize this, among which a ranking of customers and coordinator's preferences, an alternative that may be used here.

3.1.4 Experimental setup

The overall experimental setup of this second use case is very similar to the one of the long distance use case explained in sub-section 2.2.

3.2 Scenario 2.1 - Notification of late arrival at delivery points

3.2.1 Objective

Each day, delivery rounds are computed for each truck prior to their departure. Each truck can then start and begin the parcel delivery to customers. If a truck is too late for the

timely delivery of a customer, this particular destination will be removed from the round and will be replanned later, a notification being sent to the rescheduled customer. The figure 3.1 illustrates this. Customers c_1, c_2, c_3 and c_4 are planned to be delivered in a row, with a time window of half an hour. At most 45 minutes before the end of the window, materialized by the dashed circles, customers must be notified if the delivery appears to be late. When a customer can't be delivered within his time window, a replanning of the route may be done, as illustrated between customers c_2 and c_3, where a replanning tells the delivery man to skip c_2 and deliver c_4 directly and then come back to c_3. The goal is to maximize the number of customers delivered on time.

Parameters: the notification delay, a time limit from the latest possible delivery time in the window at which the customers must be notified if the truck is to arrive late, and a tolerance on this delay. The maximum lateness time, if a truck is to arrive more than this time, this particular destination may be rescheduled at end of day, and a tolerance on this maximum delay.

Typical values: A 45-minute notification delay and a 30-minute maximum lateness time, both with a 5-minute tolerance.

This sub-scenario can develop into several variants depending upon the exact behavior of the truck approaching destination.

1. The nominal variant, where the truck arrived at delivery points in time.
2. The standard late arrival variant, where the truck will be late but within acceptable gap. The customers will be notified of late delivery.
3. The very late arrival variant, where the truck cannot be at destination within the maximum lateness time. The delivery will be rescheduled at end of day and a notification will be sent to customer.

3.2.2 Classification

Below we provide an analysis of the described adaptation in the context of the modeling dimensions from appendix A:

Goals. Minimize the overall costs of position sendings, minimize the probability of missing notifications, minimize the energy consumption of the device, minimize the number of customers delivered lately.

Evolution: *Static*
In this scenario, goals do not change within the lifetime of the system.

Flexibility *Rigid*
The goals are prescriptive.

Duration *Persistent*
The goal is valid throughout the system's lifetime. However, as the truck passes from delivery points to delivery points, the delivery point to be considered passes from one to the next.

Multiplicity *Multiple goals*
In this scenario, we consider three different goals.

Dependency *Dependent*

The three first goals are conflicting, as higher frequencies minimize the probability of missing notifications, but cost more and consume more energy. Tradeoffs between them must be sought. The last one is independent.

Change. The adaptation concerns the frequency of position measurement and transmission, the positioning mode and the sleep mode of the device. It also concerns routes. It is mainly triggered when the position of the truck is approaching the delivery point. Other conditions that influence the adaptation are: the speed of the truck, the road conditions, the battery level of the positioning device.

Source *External and internal*

The main source of changes is external, the position of the truck. Others are internal.

Type *Functional and non-functional*

Positions are functional, but other sources are non-functional.

Frequency *Frequent*

As trucks approach delivery points, the changes will be made to gradually increase the frequencies. Route replanning should be less frequent, but not rare.

Anticipation *Unforeseeable (mostly)*

Positions of trucks are hardly foreseeable, as speed depends upon not only on the type of road but also from road, traffic, and weather conditions. Hence, adaptation will be triggered by positions transmitted by the truck.

Mechanism. Changes concern parameters of the positioning devices: which device to use, frequencies, mode, ..., but also routes.

Type *Parametric*

Changes concern parameters of the positioning devices: which device to use, frequencies, mode, ... or of the delivery routes.

Autonomy *Autonomous and assisted*

No outside intervention for the adaptations that concern the positioning device, but an external human decision may be required for route replanning..

Organization *Decentralized*

Adaptations are performed by the autonomic manager of the delivery management component, but has its effect on the positioning device of the truck, so the latency in the adaptation must be taken into account. Route replanning involves mainly the delivery management component, but may also affect the positioning device (to set the next delivery point and its time window).

Scope *Local*

Adaptation involves changes in the positioning device but also in the delivery management component, namely the route to be followed.

Duration *Short*

The amount of time required to reconfigure the device should be from seconds to minutes. Route replanning may take a little longer, but not very long. Reconfiguration have no impact on the availability of the system.

Timeliness *Guaranteed*

A time-bound must be observed for this kind of feedback control to be efficient.

Triggering *Event-trigger*
The main triggering conditions are event-based, by having positions matching certain conditions or by matching certain conditions at some time.

Effects. Geotracking objectives are matched but with better resource usage and lower cost.

Criticality *Mission-critical*
Although geotracking is currently done without adaptation of frequencies or fault-tolerance mechanisms, no guarantee can be put that no notification will be missed for any given precision. To have such guarantee, to some limit, adaptation is mission-critical.

Predictability *Probabilistic*
The result of actions in terms of system guarantees can only be measured as higher probabilities to match the objectives.

Overhead *Insignificant*
Parameters reconfiguration takes a small amount of time.

Resilience *Resilient*
Devices can be adapted without stopping, and the adaptation does not impair their operation, as the time to adapt is small compared to the typical time intervals between positioning instants.

3.2.3 Adaptation

In this sub-scenario, adaptations can happen to cope with the failure of the device or the energy management, as in the long distance truck tracking scenario. But to match the need of delivery point notification in case of late arrival, the frequency of the position sendings will raise gradually as the truck approaches the time limit in order to refine the estimation of arrival time and to be able to notify the customer on time.

If the truck appears to be late, the position sending frequency will be kept at a high enough level to get a good forecast of the arrival time to trigger route replanning when the estimated time exceeds the maximum lateness time. Route replanning is the major difference between the long distance truck tracking scenario and this one.

CHAPTER 4

Scenario 3 - GeoHub QoS enforcement

4.1 Context

4.1.1 Overall goals of the supporting application

Given the large-scale nature of the intended system, tens of thousands of trucks being geotracked at the same time, another important adaptation will be sought, namely the adaptation of the overall workload of the GeoHub given its current performance. As the delay between position receptions on the GeoHub and the notification of events to applications must be kept under a limit defined by the quality of service offered to customers, massive adaptations of all positioning device frequencies may be required when this delay becomes too large. This scenario will show the capability of the SALTY architecture to cope with large-scale adaptations of distributed systems.

4.1.2 GeoHub QoS enforcement use case

GeoHub QoS enforcement use case is founded on the following hypothesis:

- Every positioning device connected to the GeoHub has at least one autonomic manager component responsible for the management of its frequency. Whenever a positioning device wishes to send data to the Geohub, its autonomic manager will register itself with the GeoHub autonomic manager each time a geotracking period begins. They unregister themselves when their geotracking period ends.

- Autonomic managers of positioning devices can be interconnected in a large-scale mesh where autonomic managers are nodes connected by neighborhood relationships (knowledge of each other) used to exchange messages. This mesh can be dynamically constructed, by adding new autonomic managers when devices connect to the GeoHub, and adapted in order to minimize the number of neighbors for each autonomic manager and at the same time minimize the average path length between any two autonomic managers.

- Autonomic managers of positioning devices accept a role in the overall management of the GeoHub workload, not only to adapt the frequency of their attached positioning device, but also to participate in exchanges with other autonomic managers to reach collective decisions and coordination. In doing so, autonomic managers adopt a collaborative behavior, *i.e.* use and manage the positioning sending frequency resource in a way to maximize global optimization criteria rather than local ones.

The scenario involves two objectives:

1. **Global GeoHub workload management**, where a limit on the overall frequency of position sendings to the GeoHub must be maintained under a certain limit corresponding to Deveryware's QoS objectives. When the workload exceeds this limit, all of the connected positioning devices will be required to lower their frequency, to get a decrease in the overall workload of the GeoHub.

2. **Local frequency limits management**, where each positioning device and their autonomic managers will observe an upper bound on the frequency of its position sendings. Such limits will be adapted at run-time according to the overall workload of the GeoHub and the relative importance of the current geotracking objectives currently driving the use of this device.

4.1.3 Overall description of the adaptation scenarii

A crucial constraint in this scenario is the large-scale nature of the managed system. Indeed, the separation in two complementary business objectives of this scenario already takes into account the need for both a global and a local view of this adaptation scenario. Indeed, to go large-scale, no solution involving a centralized decision-making process can be adopted. Decentralized, heuristic schemes will be required.

For the first objective, the QoS of the GeoHub will be monitored and its autonomic manager will trigger an adaptation when the QoS becomes too low. This adaptation will consist in enforcing a decrease of the frequencies of the positioning devices currently attached to the GoeHub. To this end, the mesh of their autonomic managers can be used to broadcast a decrease frequency request, according to some protocol (to be decided at design time of the SALTY architecture).

For the second objective, individual positioning devices and their autonomic managers will adapt the frequency of position sendings corresponding to the some *scenarii* (long or short distance truck tracking), but will be limited in their increase, a limit chosen to ensure a manageable overall workload for the GeoHub. Hence, this objective complements the first by decentralizing the workload control.

4.1.4 Experimental setup

The overall experimental setup of this second use case is the same as the one of the long distance use case explained in sub-section 2.2.

4.2 Scenario 3.1 - Global workload management

4.2.1 Objective

The GeoHub is a kind of complex event processing system, running on servers connected to the Internet but also capable of emitting and receiving SMS over the GSM network. It acts as a middleware to connect geotracking applications to positioning devices of tracked mobiles, to abstract the formers from the specifics of the latters, but also to execute rules correlating positions and other data to trigger notifications to applications. GeoHub receives positions and related data from sensors on mobiles, executes each of the rules tagged by the sending device and, for each of them, when the guard of the rule is true, it executes its body, typically consisting of instructions that will send notifications to the client-side application.

As the delays between the emission of a position and the reception of the corresponding notification by the application usually needs to be bounded, the latency in the processing of data must be kept under some limit. As the delays for data transmission between devices and the GeoHub as well as between the GeoHub and application servers are in the order of seconds (using GPRS for the first), the GeoHub processing time must also be kept within the same order of magnitude.

This processing latency depends upon the quantity of computational resource dedicated to the GeoHub, and the rate of position receptions to be processed. As resources allocated to the GeoHub can only be added by large unit (servers), corresponding to a large processing capability, there is a need to manage relatively small variations in the workload when the latency reaches its limit but the workload is not large enough to justify another full server. The idea is then to ask all of the positioning devices to reduce slightly their frequencies so to reduce the overall workload and therefore the latency. But the different positioning devices are not engaged in the same business objectives when ordered to lower their frequency. Hence, a mechanism must be adopted to allow them to adapt the request to their current needs. The devices and their autonomic managers can be trusted to act "cooperatively", *i.e.* they aim at a global satisfactory solution rather than maximizing their own advantage.

Parameter: the maximum latency from position reception to notification sending, and a function allowing to compute the relative importance of each device given its current business objective.

Typical values: 2 seconds and, in the long distance truck tracking scenario, a function saying that notification of arrival at a warehouse is more important than corridor enforcement which is in turn more important than waypoint notification.

This sub-scenario can develop into several variants:

1. The nominal variant, where the latency remains under the maximum admissible value.
2. The lower complexity variant, where a (relatively) small number of devices overload the GeoHub by their individual high frequencies.
3. The higher complexity variant, where a large number of devices with relatively low frequencies yet overload the GeoHub.

In the last two variants, devices of different relative importance must be simulated to check that they adopt different modifications of their frequency.

4.2.2 Classification

Below we provide an analysis of the described adaptation in the context of the modeling dimensions from appendix A:

Goals. Keep the latency of the GeoHub under some predefined limit.

Evolution: *Static*
In this scenario, the goal does not change within the lifetime of the system.

Flexibility *Rigid*
The goal is prescriptive.

Duration *Persistent*
The goal is valid throughout the system's lifetime.

Multiplicity *Single goal*
In this scenario, we consider only one goal.

Dependency *N/A*

Change. The adaptation concerns the frequencies of position sendings for all of the positioning devices currently connected to the GeoHub.

Source *Internal*
The source of changes is internal, the latency of the GeoHub.

Type *Non-functional*
Latency is a non-functional property.

Frequency *Rare*
Pushing the server to its limit should not occur very often, and the adaptation must be powerful enough to make sure another adaptation will not be needed soon, given the extent of the required modifications.

Anticipation *Unforeseeable (mostly)*
Except for certain periods of the day where a lot of tracked vehicle are typically on the road, the changes are unforeseeable.

Mechanism. Changes concern parameters of the positioning devices, namely their position sending frequencies.

Type *Parametric*
Changes concern parameters of the positioning devices: frequencies.

Autonomy *Autonomous*
No outside intervention.

Organization *Decentralized*
Adaptations are performed by the autonomic manager of the GeoHub but in cooperation with the autonomic managers to which positioning devices are attached.

Scope *Global*
Adaptation involves changes potentially in all the positioning devices connected to the GeoHub.

Duration *Short*
Given the large-scale of the adaptation, the time required to complete it should be in the order of ten minutes.

Timeliness *Guaranteed*
A time-bound must be observed for this kind of feedback control to be efficient.

Triggering *Event-trigger*
The main triggering conditions are event-based, by having positions matching certain conditions or by matching certain conditions at some time.

Effects. Lower the latency of the GeoHub.

Criticality *Mission-critical*
Timely notifications are at the heart of the business of Deveryware.

Predictability *Probabilistic*
The result of actions in terms of system guarantees can only be measured as higher probabilities to get a lower latency.

Overhead *Insignificant*
The essential impact on the system should be the network bandwidth used to propagate the command to lower the frequency, but an appropriate multicast protocol shall maintain the required bandwidth to a relatively small level.

Resilience *Resilient*
Devices can be adapted without stopping, and the adaptation does not impair their operation, as the time to adapt is small compared to the typical time intervals between positioning instants.

4.2.3 Adaptation

The adaptation in this scenario is to modify the frequency of positioning devices. Coarsely speaking, a request to lower the frequencies by a given percentage will be issued by the autonomic manager attached to the GeoHub towards positioning devices through their own autonomic managers. Each of the device autonomic manager will have the possibility to adapt the request according to the relative importance of the device and of the current business objective served by its geotracking.

4.3 Scenario 3.2 - Local workload management

4.3.1 Objective

In the context of enforcing a global constraint on the QoS of the GeoHub, and coping with the large-scale nature of the system, a complementary approach to the first sub-scenario is to individually cap the frequency of each positioning device. Strict caps could even prevent any overloading of the GeoHub, but this would either over-constrain the positioning devices (if the caps are very low) or lead to poor resource usage (if the caps are too large and rarely completely used). Indeed, if the sum of the caps of the connected devices has to be less or equal to the maximum workload of the GeoHub, high caps would result in unused frequencies in devices requiring less than their cap. Another problem would be to impose a fixed cap to all the devices. No only different clients may have different constraints on their use of geotracking, most of the use cases presented in the SALTY project show that there can be large differences in the required frequency over time even for one device.

Hence, caps should be adapted at run-time, so that higher caps are allocated to devices facing stringent requirements from their current business objective. For example, in the long distance truck tracking scenario, not only the cap can be low when a truck is on a highway and its futures positions rather easy to predict, but one can imagine that higher caps will be allocated when tracking the arrival at a warehouse compared to notifying the passage by a waypoint.

Local workload management involves the exchange of unused frequencies between positioning devices to adapt their caps to the relative importance of their current business objectives, while maintaining constant the overall sum of the caps. Again, an important constraint put on the implementation of a solution in support to this sub-scenario is to scale to a large number of positioning devices (this rules out any centralized solution,

e.g. a central entity in charge of allocating or coordinating the allocation of frequencies to devices).

Parameters: an initial cap to the frequency of positioning devices, and a function allowing to compute the relative importance of each device given its current business objective.

Typical values: 10 position sendings per minute and, in the long distance truck tracking scenario, a function saying that notification of arrival at a warehouse is more important than corridor enforcement which is in turn more important than waypoint notification.

This sub-scenario can develop into several variants, among which:

1. The low workload variant, where most of the devices have unused frequency capacities that ease the acquisition of capacity by the requiring devices.
2. The high workload variant, where only a relatively small number of devices have unused capacities, which makes difficult the acquisition of capacity by the requiring devices.
3. The saturated workload variant, where the whole system is overloaded so that the requiring devices cannot be satisfied.

4.3.2 Classification

Below we provide an analysis of the described adaptation in the context of the modeling dimensions from appendix A:

Goals. Adapt locally the maximum frequency at which each device can send positions in order not to overload the GeoHub.

Evolution: *Static*
In this scenario, the goal does not change within the lifetime of the system.

Flexibility *Rigid*
The goal is prescriptive.

Duration *Persistent*
The goal is valid throughout the system's lifetime.

Multiplicity *Single goal*
In this scenario, we consider only one goal.

Dependency *N/A*

Change. The adaptation concerns the frequencies of position sendings for each of the positioning devices currently connected to the GeoHub.

Source *Internal*
The source of changes is internal, the frequency currently required by the positioning devices and request coming from other devices to get unused capacities when they need to higher their own maximum frequency.

Type *Functional*
Position sending frequency is a functional property.

Frequency *Frequent*
As the caps to frequencies should be kept relatively near to the individual currently used frequencies, the adaptation should happen relatively often. However, the mean difference between the caps and the used frequencies is an crucial parameter to tune the system, as it represents in some sense the degree at which the system will be able to locally adapt used frequencies to the need of applications. When a device has to require unused frequencies from other devices, the local adaptation can become more and more global to find unused capacities somewhere.

Anticipation *Unforeseeable*
As for the used frequencies themselves, changes in the maximum frequencies are unforeseeable, mostly, even though business objectives that drive the adaptation are known in advance.

Mechanism. Changes concern parameters of the positioning devices, namely their maximum position sending frequencies.

Type *Parametric*
Changes concern parameters of the positioning devices: the maximum frequencies.

Autonomy *Autonomous*
No outside intervention.

Organization *Decentralized*
Adaptations are performed by the autonomic managers of the requiring device but in cooperation with the autonomic managers of other positioning devices.

Scope *Local (mostly)*
Adaptation involves changes in a small number of the positioning device autonomic managers.

Duration *Short*
The time required to complete the adaptation should be kept in the order of minutes.

Timeliness *Guaranteed*
A time-bound must be observed for this kind of feedback control to be efficient.

Triggering *Event-trigger*
The main triggering conditions are event-based.

Effects. Enhancing the probability for applications to match their business objectives, while maintaining the quality of service of the GeoHub over its target.

Criticality *Mission-critical*
At the heart of Deveryware's business model.

Predictability *Probabilistic*
The result of actions in terms of system guarantees can only be measured as higher probabilities to get a lower latency.

Overhead *Insignificant*
The chosen implementation must strive for an insignificant overhead.

Resilience *Resilient*
Changes in the maximum frequencies of device can be done without impairing their functioning.

4.3.3 Adaptation

The adaptation incurs:

- finding unused capacity, by adapting the maximum frequency(ies) of a(some) device(s),
- adapt the maximum frequency for the device,
- adapt the current frequency of the device.

CHAPTER 5

Scenario 4 - Decision-making Modeling at Design-time

This scenario is of a different nature compared to the previous ones. One of the goals of the SALTY project is to build an interactive tool, the decision-making modeling at design-time tool, to help non-specialist end-users in eliciting their business objectives and criteria for adaptation for their geotracking applications. As such a tool cannot be fully general, the geotracking use case will provide us with a first context for design exploration. Generality of the tool will be sought through the use of an adaptable database for parameters such as dictionaries, model-driven interaction patterns (question/answer, menu, etc.), etc. Figure 5.2 shows a general scheme from the interactive tool to the positioning devices inside the trucks and back.

5.1 Context

Deveryware offers to its customers a GeoHub configuration tool called DeveryLoc. Using this tool, users can declare alerts, defining the positioning devices to be tracked, frequency of position transmission and events to be notified along with their triggering conditions. DeveryLoc leverages a set of predefined alert types, offering users standard notifications patterns such as entering or leaving a geographical locus. All the parameters the user has to enter to configure these alert types are linked either to a duration (frequency or delay in seconds), or to a distance (in meters), *i.e.* only very precise values can be entered. Unfortunately, to date, users rarely have the necessary knowledge to provide the GeoHub with good enough information for the geotracking to be efficient. In face of a lot of technical parameters to define, users tend to fix them arbitrarily and more or less update them after some experience becomes available from the running of the application, a lengthy and costly process.

The first goal of the design-time decision-making modeling tool is to abstract users away from these precise and low-level technical parameters. A second goal is to add the configuration of adaptations to be applied to alerts. Finally, the tool will strive for a qualitative assessment of parameters rather than precise numerical values. The tool, through menus and natural language exchanges, will help users to elicit their higher level business-oriented goals, both at the application level (*e.g.* notify the application when a truck is at approximately thirty minutes from its delivery warehouse) and at the adaptation one (*e.g.* minimize the number of position transmissions while making sure that the notifications happen within a time frame according to the meaning of "approximately thirty minutes"). As several criteria may come into play when allocating position transmission frequencies, users may need to express (qualitatively) their preferences in case a trade-off is needed (*e.g.* I prefer a better precision on warehouse delivery notifications than on waypoint notifications, but if the battery is low, I prefer to keep the positioning

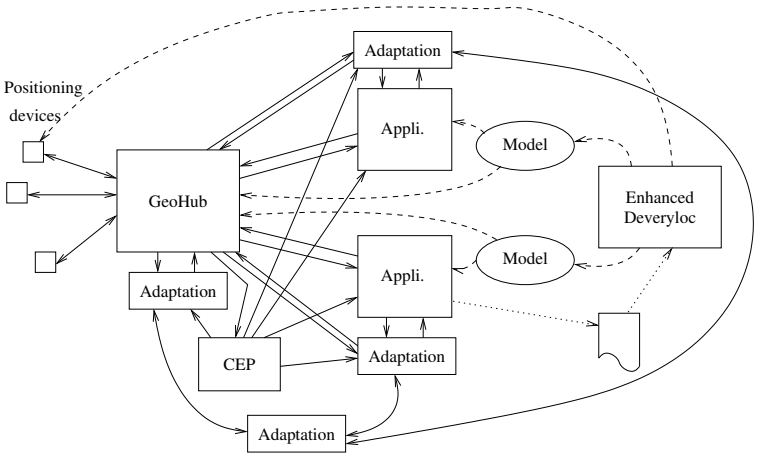


Figure 5.1: The various components of the global tool.

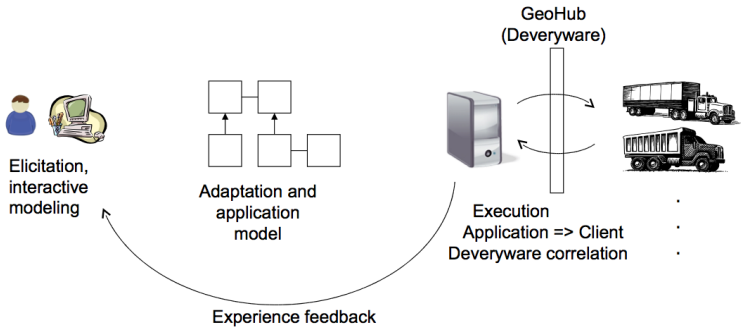


Figure 5.2: General scheme from the human-machine interface to the trucks.

device functioning as long as possible rather than a better precision on notifications).

Although developed in the context of SALTY, the tool will strive for as much generality as possible, to be adaptable to other adaptive application contexts (such as the companion middleware use cases).

Thus, the configuration tool DeveryLoc will be enhanced to deal with a larger role illustrated in Figure 5.1. In this figure, the plain arrows represent position sending by devices, alerts and/or notifications sending by the GeoHub and other events used at runtime to execute and adapt the applications. The "Enhanced DeveryLoc", or E-DeveryLoc, is a design-time tool that will provide for an enhanced HMI, allowing end-users to build models of applications and their adaptations, and then to configure the GeoHub accordingly.

Business-specific applications (denoted "Appli." in the figure) are partly generated by the E-DeveryLoc through a model. Once in place, and completed by the business-specific code on the client-side, they will execute and undergo adaptations. Such adaptations will use positions sent by the positioning devices and notifications made by the GeoHub, but also context-based information (*i.e.* weather forecasts, traffic jams, etc.). A CEP, completing the GeoHub, will handle the latter kinds of data.

Business-specific applications will also provide logs, shown on the right side of the figure, that shall be exploited by end-users, through the E-DeveryLoc again, to modify and adapt (a longer-term vision of this) their application given their operation on the

field. The figure 5.2 insists on this feature of the E-DeveryLoc, with the ”experience feedback” arrow illustrating this longer-term life-cycle management of applications.

5.2 Overall description of the tool

The decision-making modeling at design-time tool aims at bridging the gap between client-side applications and the GeoHub, attaching to the latter positioning devices of the client. Programmers using the tool may adopt one to all of the three following roles:

- an *expert role*, which consists in defining new type of alerts and new types of adaptations, their parameters, the domains for the values of these parameters, including qualitative ones;
- an *application designer role*, which consists in using the predefined alert and adaptation types to put together the necessary notification rules on the GeoHub for his application;
- a *configurator role*, which consists in giving all the necessary parameters to run the alerts, such as the precise device identifiers, the periods of time during which the geotracking must occur, etc.

The third role corresponds mostly to the current DeveryLoc tool. The two first ones concern the design and use of new types of alerts and adaptations. Without dealing immediately with the tool design issues, it nevertheless shall take into account the following requirements:

- The tool will be Web-based in order to ease its access for Deveryware’s clients. However, its internal architecture should provide for an easy adaptation to other interfaces and contexts, such as an integration into development tools like Eclipse.
- It will be qualitative assessment oriented (linguistic approaches) to overcome the problems raised above, *i.e.* the replacement of precise values (the parameters the user has to enter) by qualitative or linguistic ones. Indeed, we know that humans deal with words and usually not numbers in their everyday lives, especially when reasoning. Human reasoning is perception-based. For example, one can say: ”it seems to be *rather cold* this morning: I’m going to wear a thick pullover” instead of ”it is $12.25°C$ this morning: I’m going to wear a .5cm-thick lambswool pullover”. In our context, we shall use words as *far, very near, quickly*, etc. Several models exist in the literature to deal with this problem, such as fuzzy and qualitative models, all these belonging to the *Computing with Words* paradigm. In these models, the objects of computation are words and the principle is to model these words through membership functions or scales or 2-tuples, etc. Operations are then made on them, aggregating or modifying them.
- The tool shall use natural language question/answer exchanges whenever possible: to enhance the HMI and to make it more flexible, we propose to add a non context-free natural language interface. The advantage of HMI is conclusive proof since user-friendly interface means ”user-understanding” interface. We will first proceed to a basic discourse analysis which deals with the main levels of linguistics, such as morphology (”a big” or ”a bag” or ”a bug”, etc.), syntax (word order and/or lack of words), semantics (a truck thinks, a man thinks) and some pragmatics (context

contributes to semantics), which is a challenge to artificial intelligence techniques, for example ”the truck is in a difficult situation” that can be interpreted as ”the truck is caught in a traffic jam” or ”the truck is itself blocking the traffic if it had an accident” or ”the truck has been arrested at the border and there is a problem to clear through customs” or . . . Another advantage of this HMI would be to learn from a logic pattern (grammatical or syntactic model) how to build new interfaces (as a writer who uses a common dictionary to propose a new novel).

- The tool should produce a description of the notifications and adaptations in the form of a model, conforming to a predefined meta-model, enabling the generation of the code[1] to be deployed on the software architecture (application components, adaptation components, GeoHub and positioning devices), including:
  - notifications, and their triggers
  - tolerance on triggers
  - adaptation criteria
  - preferences among criteria

5.3 Scenario 4 - Arrival at warehouse notification definition

This section describe the use of the E-DeveryLoc tool through one of the sub-scenario from the long distance truck tracking application: the arrival at warehouse notification. The use case is presented in three parts corresponding to the three roles identified above: expert, application designer and configurator roles.

5.3.1 Expert role

As an example, consider an expert user whose business objectives are to track his truck fleet going to warehouses. He will have to choose which notifications he wants and their associated triggers, what tolerance he shall accept on triggers, the kind of adaptations he wants to be applied when geotracking for warehouse arrival notifications, what he prefers among criteria in case trade-offs must be made.

Through a series of exchanges with the tool, this user will define a new type of alert that will be used by the GeoHub to notify the user application when a truck arrives at a warehouse. These exchanges will take several forms: choices in menus, natural language questions/answers, etc. The resulting model should take the form of new alert types, which representation in XML shall resemble to the XML file shown in Figures 5.3 and 5.4.

In Figure 5.3, a geotracking model is defined, with its own URI, from a metamodel that will be defined in details as a result of the SALTY project. The model consists of alert type definitions. An alert type has a name and a category (to ease their use afterwards). It has three kinds of parameters:

1. **application parameters**, that correspond to what need to be defined to use this type of alert in a particular application;

[1] The code generation *per se* is not part of the SALTY project, but the tool should be implemented to pave the way to such work. Moreover, these models will be used in use cases to help in by translating them by hands into code.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<geotracking-model
    uri="http://www.deveryware.com/model/TruckTracking"
    xmlns="http://move.lip6.upmc.fr/geotracking-model/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <alertType name="ArrivalAtWarehouseNotification" category="ArrivalAtADestination">
    <application-parameters>
      <param name="Delay" type="xsd:time"/>
      <param name="DelayTolerance" type="QualitativeTimeDelayTolerance"/>
    </application-parameters>
    <configuration-parameters>
      <param name="PositioningDevice" type="PositionDeviceType"/>
      <param name="StartTime" type="xsd:dateTime"/>
      <param name="EndTime" type="xsd:dateTime"/>
    </configuration-parameters>
    <runtime-parameters>
      <param name="CurrentPosition" type="GPSValue"/>
      <param name="WarehouseLocation" type="GPSValue"/>
    </runtime-parameters>
    <trigger type="FIS">
      <param name="Pos" type="GPSValue"/>
      <fis name="triggerArrivalNotification">
        <fuzzyRule>
          <antecedent type="QualitativeTimeTolerance">
            <lingValue>
              <2tuple term="closeTo" index="1" symbolicTranslation="0.0"/>
            </lingValue>
          </antecedent>
          <consequent type="TriggeringLevel">
            <lingValue>
              <two-tuple term="high" index="4" symbolicTranslation="0.25"/>
            </lingValue>
          </consequent>
        </fuzzyRule>
        ...
      </fis>
      <infer fisName="triggerArrivalNotification">
        <expression op="minus">
          <applyFunction name="estimateTravelTime">
            <with-param name="from" value="$CurrentPosition"/>
            <with-param name="to" value="$WarehouseLocation"/>
          </applyFunction>
          <value-of exp="$Delay"/>
        </expression>
      </infer>
    </trigger>
    <emittedEvent type="ArrivalAtWarehouseNotificationEvent"/>
    <description lang="en-En">
      When a truck can be predicted to arrive at the specified warehouse
      at the given Delay, the specified event is emitted towards the
      application.
    </description>
  </alertType>
```

Figure 5.3: Application model captured through the E-DeveryLoc tool.

```
  <dataType name="QualitativeTimeTolerance">
    <lingVariable xmlns="http://www.univ-paris8.fr/schema/lingVar">
      ...
      <lingValue name="closeTo"> ... </lingValue>
      ...
    </lingVariable>
  </dataType>
  <dataType name="TriggeringLevel">
    <lingVariable xmlns="http://www.univ-paris8.fr/schema/lingVar">
      ...
      <lingValue name="high"> ... </lingValue>
      ...
    </lingVariable>
  </dataType>
  <dataType name="ArrivalAtWarehouseNotificationEvent">
    <data name="TruckId"  type="xsd:token"/>
    <data name="occurenceTime" type="xsd:dateTime"/>
    <data name="WareHouseLocation" type="GPSValue"/>
  </dataType>
</geotracking-model>
```

Figure 5.4: Application model captured through the E-DeveryLoc tool (cont.).

2. **configuration parameters**, that correspond to what need to be defined to execute the alert on the GeoHub (at least an identification of the positioning device, as well as the start and finish times of the activation of the alert);

3. **runtime parameters**, that correspond to values that will be known only during program execution and supplied by the GeoHub.

To fully define parameters at this stage, datatypes may need to be supplied. The figure 5.4 show data types defined in this model, the two first being qualitative ones described next.

Qualitative modeling

Especially, when qualitative parameters or other data are introduced into alert types, their corresponding linguistic representation will need to be provided. The tool will also help users in defining such qualitative values, through exchanges that will require not only to give names (terms) to represent qualitative values, but also their corresponding fuzzy representation. As another result to the SALTY project, an XML meta-model will be proposed to define the qualitative model provided by the user in terms of linguistic variables, modifiers, etc.

The figure 5.5 gives a first example of such a model with a Relax NG grammar in the compact syntax. Linguistic variables are defined with a name, a support set (*e.g.* time durations) and a set of terms representing the qualitative values themselves. Qualitative terms are linked to fuzzy subsets, in support to a fuzzy qualitative reasoning. This first grammar puts forward the common use of trapezoid and triangular fuzzy subsets. After their definition, linguistic values can be used in the geotracking models as references to the term set of a linguistic variable. The figure also puts forward our intention to use the 2-tuple models of representation for linguistic values, as said in the SALTY project definition.

```
# File: "LingVariable.rnc"
# A RELAX NG grammar in the compact syntax.
# Description : Qualitative model with linguistic variables.
# Version : 1.0

defaultnamespace = "http://www.univ-paris8.fr/schema/lingVar"

# Linguistic variables
LingVariable  = element lingVariable { Name, SupportSet, LingValueDefs }
Name          = attribute name { NSName }
SupportSet    = element supportSet { Name }
# Linguistic value definitions
LingValueDefs = element lingValues { LingValueDef+ }
LingValueDef  = element lingValueDef { Term, Index?, (FuzzySubset | Crisp) }
Term          = attribute term { NSName }
Index         = attribute index { xsd:int }
Crisp         = element value { xsd:double }
FuzzySubset   = element FuzzySubset  { TrapezoidFS | TriangularFS | AnyFS }
TrapezoidFS   = element trapezoidFS  { Point, Point, Point, Point }
TriangularFS  = element triangularFS { Point, Point, Point }
AnyFS         = element anyFS        { Point+ }
Point         = element point {
    attribute abcissa  { xsd:double },
    attribute ordinate { xsd:double { minInclusive="0.0" maxInclusive="1.0"} }
}
# Linguistic value references
LingValue     = element lingValue { FuzzySubset | 2tuple | Crisp }
2tuple        = element 2tuple    { Term, Index, SymbolicTranslation }
SymbolicTranslation = element symbolicTranslation {
    xsd:double { minInclusive="-0.5" maxExclusive="0.5" }
}
```

Figure 5.5: A first Relax NG schema for linguistic variables.

Adaptation model

After defining alert types, the expert can define their adaptations types. Adaptations aim at providing policies to decide adaptation actions upon the state of a process. In doing so, it applies decision criteria, aggregated through the use of user preferences. The figure 5.6 shows an XML excerpt providing these different adaptation parameters and criteria for the major type of adaptation sought in our *scenarii*, the position sending frequency adaptation. This adaptation relates the current position and the current frequency to a new frequency to be applied for the next period of time. Three criteria are defined: probability to get a position matching the need of the geotracking objective, battery usage and total transmission cost. As a multi-criteria decision making problem, user preferences are needed to aggregate the criteria into one unique utility value to be optimized. The next paragraph explains how such preferences will be expressed in the SALTY project.

Preference modeling

For preferences, the SALTY project will primarily use the LCP-nets (Linguistic Conditional Preferences Networks) formalism, as defined in [1]. To be integrated in the scenario, the figure 5.7 shows a graphical representation of the following preference of the user for the precision of its notifications: ”I prefer more precise warehouse arrival notifications over waypoint passage notifications, but if the battery level is low I prefer to

```
<?xml version="1.0" encoding="iso-8859-1"?>
<adaptationTypes
    uri="http://www.deveryware.com/model/TruckTrackingAdaptations"
    xmlns="http://move.lip6.upmc.fr/adaptation-meta-model/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <adaptationType name="PositionTransmissionFrequencyAdaptation">
    <state name="currentPosition" type="GPSValue"/>
    <state name="transmissionFrequency" type="xsd:float"/>
    <action name="newTransmissionFrequency" type="xsd:float"/>
    <criterion name="getPositionProbability" relation-semantics="increasing">
      <function>
        <param name="targetPosition" type="GPSValue"/>
        <param name="targetRadius" type="xsd:decimal"/>
        <expression>
          if (withinRadius(nextPosition(currentPosition,
                                        newTransmissionFrequency),
                           targetPosition, targetRadius)) then
              1
          else if (beforeRadius(predictedPosition(currentPosition,
                                                  newTransmissionFrequency),
                                targetPosition, targetRadius)) then
              1
          else 0
        </expression>
      </function>
    </criterion>
    <criterion name="batteryUsage" relation-semantics="decreasing">
      <function>
        <param name="consumptionPerTransmission" type="xsd:float"/>
        <expression><value-of exp="$consumptionPerTransmission"\></expression>
      </function>
    </criterion>
    <criterion name="totalTransmissionCost" relation-semantics="decreasing">
      <function>
        <param name="costPerTransmission" type="xsd:float"/>
        <expression>costPerTransmission</expression>
      </function>
    </criterion>
    <aggregation>
      <preferences>
        <lcp-net>...</lcp-net>
      </preferences>
    </aggregation>
  </adaptationType>
  ...
</adaptationTypes>
```

Figure 5.6: Adaptation model for the truck tracking scenario.

save battery levels rather than precision in the notifications." In the figure, B represents the battery level, PW the precision on warehouse notifications and PP the precision on waypoint notifications.

The LCP-net reads as follows. In the center, the net itself shows that battery level B is always preferred to precisions with conditional preference arcs from the node B to both the precision in warehouse arrival notifications PW and the precision in waypoint passage notifications PP. The mid-arrow going from PW to PP is an i-arc expressing the unconditional preference of the user for PW over PP.

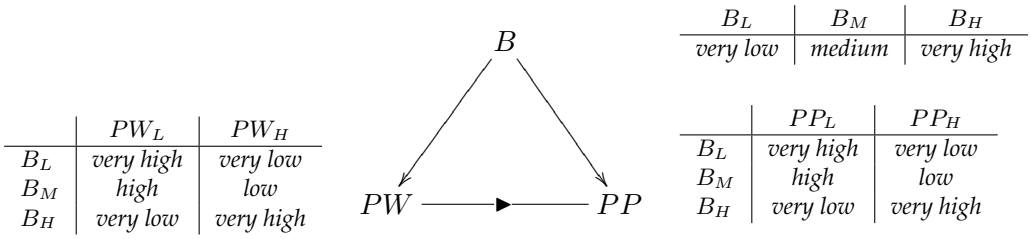


Figure 5.7: A user preference example using LCP-nets.

The tables provide the local utilities for the different assignments to the variables. The table besides B says that a qualitative local utility *very low* is given to low battery level B_L, *medium* to a medium battery level B_M and *very high* to a high battery level B_H.

The two tables attached to the PW and PP nodes are conditional, to express the fact that the preferences of the user on values of these variable depends upon the value of B. Both tables show a reversal in the utilities between the low and the high battery levels that expresses the fact that the user prefer lower precisions when the battery level is low, as this will lower the battery usage by requiring less position sendings.

An XML syntax has been defined to represent LCP-nets. This syntax will be used to add LCP-nets to adaptation models.

In the SALTY project, Thales also proposes an alternative way to express preferences, an approach based on Choquet integrals. This alternative will be compared to LCP-nets, and will be taken into account by the decision making component of the MAPE-K loop.

5.3.2 Application designer role

The application designer role consists in using the E-DeveryLoc tool, and the set of alert types defined by experts, to define alerts for his application. In this role, the E-DeveryLoc will organize he exchanges with the application designers in such a way to select the necessary alerts from its alert types library, and then make them provide the application parameters for each alert as a choice among their possible values, as defined by the expert in his data types. Similarly, from existing adaptation models, the application designer will be able to select among them the ones that apply to his application.

After the exchanges, application alert descriptions will be generated in the form of an XML description of the alerts, as exemplified in Figure 5.8. In this scenario, the application designer defines an alert for the arrival at the Nice warehouse. He provides the location of the warehouse, as GPS coordinates, as well as the notification delay and the qualitative tolerance he puts on this delay.

5.3.3 Configurator role

The configurator role consists in using the E-DeveryLoc tool and an application as defined by the application designer to configure the geotracking at truck start time. Here, exchanges aim at providing the configuration parameters of the alert types. The result will be a configuration file as exemplified in Figure 5.9. In this scenario, the configurator provides the positioning device information, according to the mobile model shown in Figure 5.10. He also provides starting and finishing times for the geotracking.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<geotracking-application
    uri="http://www.deveryware.com/application/LongDistanceTruckTracking"
    applicationModel="http://www.deveryware.com/model/TruckTracking"
    adaptationModel="http://www.deveryware.com/model/TruckTrackingAdaptations"
    xmlns="http://move.lip6.upmc.fr/geotracking-program-model/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <alert name="ArrivalAtNiceWN" type="ArrivalAtWarehouseNotification">
    <with-param name="WarehouseLocation">
      <gpsValue lat="43.7" long="7.26"/>
    </with-param>
    <with-param name="Delay" value="00:30:00"/>
    <param name="DelayTolerance" value="closeTo"/>
  </alert>
  ...
  <adaptation name="frequencyForArrivalAdaptation"
              type="PositionTransmissionFrequencyAdaptation"/>
  ...
</geotracking-application>
```

Figure 5.8: Application description captured through the E-DeveryLoc tool.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<geotracking-configuration
    uri="http://www.deveryware.com/configuration/LongDistanceTruckTrackingConfig"
    application="http://www.deveryware.com/application/LongDistanceTruckTracking"
    type="http://www.deveryware.com/application/LongDistanceTruckTracking"
    xmlns="http://move.lip6.upmc.fr/geotracking-model/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <alertConfiguration alert="ArrivalAtNiceWarehouseNotification">
    <with-param name="PositioningDevice">
      <mobile name="truck1234" xmlns="http://www.deveryware.fr/schema/mobile"
        <phoneNumber>06123456</phoneNumber>
      </mobile>
    </with-param>
    <with-param name="StartTime" type="2010-10-26T08:00:00"/>
    <with-param name="EndTime" type="2010-10-31T18:00:00"/>
  </alertConfiguration>
  ...
</geotracking-configuration>
```

Figure 5.9: Application configuration captured through the E-DeveryLoc tool.

5.4 Experimental setup

The experimental setup for this scenario involves making use of the E-DeveryLoc tool to define alert types and alerts. All of the geotracking objectives and adaptations from the previous *scenarii* will be used to this end. By the end of the project, experiments with Deverware personnel using the tool will be made in order to assess the usability of the tool, and its capability to handle more or less unpredictable, though domain-specific, exchanges in natural language.

```
# File: "Mobile.rnc"
# A RELAX NG grammar in the compact syntax.
# Description : model of mobile positioning devices.
# Version : 1.0

defaultnamespace = "http://www.deveryware.fr/schema/mobile"

Mobile = element mobile { Name, PhoneNumber, GPSid?, SynchroInterval?}
Name = element name { text }
PhoneNumber = element phoneNumber { text }
GPSid = element gpsid { text }
SynchroInterval = element synchroInterval { text }
```

Figure 5.10: A first Relax NG schema to define mobile information.

Bibliography

[1] Pierre Châtel, Isis Truck, and Jacques Malenfant. LCP-nets: A linguistic approach for non-functional preferences in a semantic SOA environment. *Journal of Universal Computer Science*, 16(1):198–217, 2010. `http://www.jucs.org/jucs_16_1/lcp_nets_a_linguistic`.